

AD-A115 565

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHO0--ETC F/G 9/2
MULTI-CLASS ANALYTICAL MODELS OF THE DECSYSTEM-10 JOB-SWAPPING --ETC(U)
DEC 81 M H COX
UNCLASSIFIED AFIT/GOR/MA/B1D-4

NL





FILE COPY

- AD A115565

AFIT/GOR/MA/81D-4

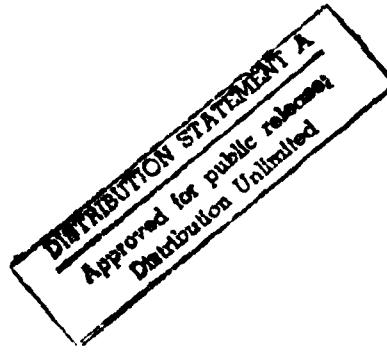
D

MULTI-CLASS ANALYTICAL MODELS OF THE
DECSYSTEM-10 JOB-SWAPPING BEHAVIOR

THESIS

AFIT/GOR/MA/81D-4

Michael H. Cox
Captain USAF



MULTI-CLASS ANALYTICAL MODELS
OF THE DECSYSTEM-10 JOB-SWAPPING BEHAVIOR

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Michael H. Cox
Capt USAF

Graduate Operations Research

December 1981

Accession For	NTIS GRA&I	<input checked="" type="checkbox"/>
	DTIC TAB	<input type="checkbox"/>
Unannounced	Announced	<input type="checkbox"/>
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or	Special
A		



Approved for public release; distribution unlimited.

Preface

This report is the result of an effort to further develop computer performance evaluation (CPE) tools for personnel at the Air Force Avionics Laboratory, Wright-Patterson AFB, OH. A previously developed FORTRAN program to solve multi-class closed queueing networks was modified to include an approximation algorithm to improve the accuracy of modeling job swapping.

Not only was this an academic learning experience, I feel I have gained some insight into the problems of getting support for a worthwhile project from the management structure. Sometimes this can be the most difficult obstacle in a research project. Also, I have learned a few things about myself in the process. I am hopefull that I will use this knowledge to take advantage of my strengths and improve on some of my weaknesses.

First, I would like to thank my thesis advisor, Ltc James Bexfield, for his support in this thesis effort. His encouragement and guidance were only exceeded by his patience.

Also of great moral support were my parents. Their many cards and letters and long-distance phone calls were always full of encouragement.

I would like to thank all of my classmates in both the GOR-81D and GCS-81D for their help through the year with the many problem sets, term projects, and labs. A special thanks goes to Ken Bauer for his artwork on the plots in this thesis. More importantly, his light-hearted humor and warm smile were always there to cheer me up. May you never lose your tremendous concern for others.

Finally, I would like to thank a small software company called Mark of the Unicorn for developing Scribble, the best text formatter for a CP/M home computer I've ever seen, and for their patient help when I needed it.

Michael H. Cox

Table of Contents

Preface	ii
List of Figures	viii
List of Tables	ix
Abstract	x
Chapter 1 Introduction	1
1.1 Computer Performance Evaluation (CPE)	1
1.1.1 Need for CPE	1
1.1.2 CPE Techniques	2
1.1.3 Development of Analytical Models in CPE	5
1.1.3.1 Single Queue, Infinite Population Model	5
1.1.3.2 Finite Population Model	6
1.1.3.3 Central Server Model	7
1.1.3.4 Classical Swapping Model	9
1.1.3.5 Chen's Swapping Model	10
1.2 Problem Statement	11
1.2.1 Motivation for Research Objectives	11
1.2.2 Background	12
1.2.3 Research Objectives	13
1.2.3.1 Primary Objective	13
1.2.3.2 Specific Objectives	13
1.2.4 Scope of the Thesis	14
Chapter 2 Operating Systems and CPE	1
2.1 Operating System Objectives	2
2.2 Resource Manager Functions	2
2.3 The DECsystem-10 Operating System: TOPS-10 Monitor	5
2.3.1 Monitor Programs	5
2.3.2 Job States	7

Table of Contents (cont.)

2.3.3 Monitor Queueing Structure	11
2.3.3.1 In-core Versus Out-core Chains	11
2.3.3.2 Processor Queues	11
2.3.3.3 Long-term Wait Queues	12
2.3.3.4 Processor Queue Time Slices	
	14
2.4 Chapter Summary	17
Chapter 3 Analytical Modeling	1
3.1 Early Queueing Network Models	1
3.2 Multi-class Queueing Network Model	3
3.2.1 Variables in the Model	4
3.2.2 Open Network Outside Arrival Processes	5
3.2.3 Service Time Distributions	6
3.2.4 State-dependent Service Rates	
	7
3.2.5 Service Center Types	7
3.2.6 States of the Model	9
3.2.6.1 Type 1 Service Center	9
3.2.6.2 Type 2 and Type 3 Service Centers	10
3.2.6.3 Type 4 Service Centers	10
3.2.7 Calculating Equilibrium State Probabilities	11
3.2.7.1 Balance Equations	11
3.2.7.2 Product Form Solution	12
3.2.8 Example Problem	16
3.2.8.1 The Balance Equations	17
3.2.8.2 The Product Form Solution	24
3.3 Chen's Swapping Model	26
3.3.1 Variables in the Model	27

Table of Contents (cont.)

3.3.1.1 Derivation of P_{14} and P_{12}	30
3.3.2 Program Swapping Behavior	32
3.3.2.1 Derivation of P_{43}	32
3.3.2.2 Derivation of P_{13}	34
3.3.2.3 Approximation Algorithm	36
3.4 Chapter Summary	37
Chapter 4 Computer Implementation	1
4.1 The McKenzie Program	2
4.1.1 Program Capabilities	2
4.1.2 Program Structure	3
4.1.3 Program Inputs	6
4.2 The Chen Modification	6
4.2.1 Program Structure	6
4.2.2 Program Inputs	8
4.3 Programming Notes for Future Modification	11
4.4 Chapter Summary	13
Chapter 5 Analytical Modeling Results	1
5.1 Modeling Interactive/Batch Workloads	1
5.2 DECsystem-10 System Configuration Parameters	2
5.2.1 Hardware Parameters	2
5.2.2 Workload Parameters	4
5.2.3 DECsystem-10 Swapping Model	6
5.3 Model Comparisons	8

Table of Contents (cont.)

5.3.1 Definition of Performance Measures	8
5.3.2 Tabular Results	9
5.3.3 Chen's Versus the Classical Swapping Model	12
5.3.3.1 Probability Structure	12
5.3.3.2 Performance Predictions	15
5.3.4 Chen's Swapping Model: Multi-class Versus Single-class	19
5.4 Chapter Summary	24
Chapter 6 Conclusions and Recommendations	1
6.1 Conclusions	1
6.2 Recommendations for Future Research	2
Bibliography	
Vita	
Appendix	A-1

List of Figures

<u>Figure</u>		<u>Page</u>
1-1	Spectrum of Computer Modeling Techniques	1-4
1-2	Single-resource Queueing Model	1-6
1-3	Finite Population Model	1-7
1-4	Central Server Model	1-8
1-5	Classical Swapping Model	1-10
2-1	Monitor Cycle Programs	2-8
2-2	Job State Transitions	2-10
3-1	Method of Stages	3-6
3-2	State Transition Diagram	3-18
3-3	Global Balance Equations	3-19
3-4	Local Balance Equations	3-21
3-5	Chen's Swapping Model	3-29
4-1	McKenzie Program Flow Chart	4-5
4-2	New Program Flow Chart	4-9
5-1	DECsystem-10 Job-Swapping Model	5-7
5-2	Probability Structure (Two Classes)	5-13
5-3	Two Classes	5-16
5-4	Chen's Model	5-21

List of Tables

<u>Table</u>		<u>Page</u>
3-1	Transition Matrix and Service Rates	3-16
3-2	Example Calculations	3-25
3-3	Probability Transition Matrix for Chen's Swapping Model	3-31
5-1	System Configuration Parameters	5-3
5-2	Classical Swapping Model Results	5-9
5-3	Chen's Swapping Model Performance Results	5-10
5-4	Important Probability Transition Matrix Values	5-11

Abstract

→ An improved model of the DECsystem-10 job-swapping behavior was developed. This model combines a previously developed closed queueing network model with a job-swapping model developed by Chen, (Ref 5). Chen's swapping model provides an approximate solution to a network queueing model with a state-dependent probability transition.

This combined model is then tested on a hypothetical, though realistic workload containing both interactive and batch jobs. The two classes of jobs are treated first as separate classes, as one class having the weighted average job characteristics of both classes, and as one class having just the interactive job characteristics. The results of these experiments and a comparison between Chen's swapping model and the classical are presented.

The results of the experiment indicate that it is important to model multiple classes for systems which have a significant amount of batch activity. Also, Chen's swapping model provides a more realistic model of job-swapping behavior for the DECsystem-10. Therefore, combining the

multi-class model with Chen's swapping model improves the modeling accuracy for the DECsystem-10. Recommendations for extensions to this multi-class Chen model are also discussed.

Chapter 1

Introduction

1.1 Computer Performance Evaluation (CPE)

1.1.1 Need for CPE

In recent years, science and engineering have become more concerned with the economic aspects in their fields due to increased costs of high technology and tighter fiscal budgets. Great attention has been given to the development and refinement of techniques which help predict behavior of systems and thus yield insights into what cost-performance tradeoffs can be made (Ref 23:1).

Any system which is in the process of being designed, procured, or modified must satisfy certain predetermined performance specifications. This is especially true in computer engineering where system specifications, reference manuals, and user guides abound. Designers and engineers use design and performance evaluation prediction techniques to obtain systems which meet these specifications.

Introduction

Prospective users of a system use these techniques to determine which combination of subsystems of components comes closest to matching their requirements, given certain cost constraints. Current users of a system use these same evaluation techniques and tools to help make decisions about existing systems concerning system upgrades and additions. Therefore, performance evaluation is needed at all stages of the life cycle of a computer product. (Refs 9:1,23:1-2).

Computer engineering is a fairly recent development and therefore performance evaluation is less developed than older branches of engineering, but definitely is not less important . The products of computer engineering (central processing units (CPU's), memory, printers, card readers, tape drives, disk drives, etc.) are primarily designed to perform certain functions related to the processing of information. How well these systems execute their tasks is a matter of tremendous technical, economic, and military importance to the Air Force (Ref 9:3)

1.1.2 CPE Techniques

Techniques for evaluating a computer system can be divided into two categories: measurement techniques and modeling techniques. Measurement techniques involve using

Introduction

performance monitors to detect and analyze system events. These monitors can be hardware oriented or software oriented, and their use with benchmark programs is called benchmarking. Modeling techniques, on the other hand, do not directly involve the system in question, but deal with a conceptual representation of the system (Refs 1, 9, 29).

These modeling techniques can be divided into four categories (Ref 1):

1. Rules of thumb, e.g. CPU utilization should not exceed 35 percent for on-line applications or 40 percent for batch applications.
2. Linear projection, e.g. "Computer usage doubled in the last year, so it will double again next year."
3. Analytical queueing models.
4. Simulation models.

Another CPE technique discussed by Svobodova (Ref 29:48-49) and more extensively by Sanabria (Ref 25) is empirical modeling. This technique combines the linear projection technique with benchmarking. Measurement data are collected from the real computer system and regression or some other curve-fitting technique is used to develop performance prediction curves.

Introduction

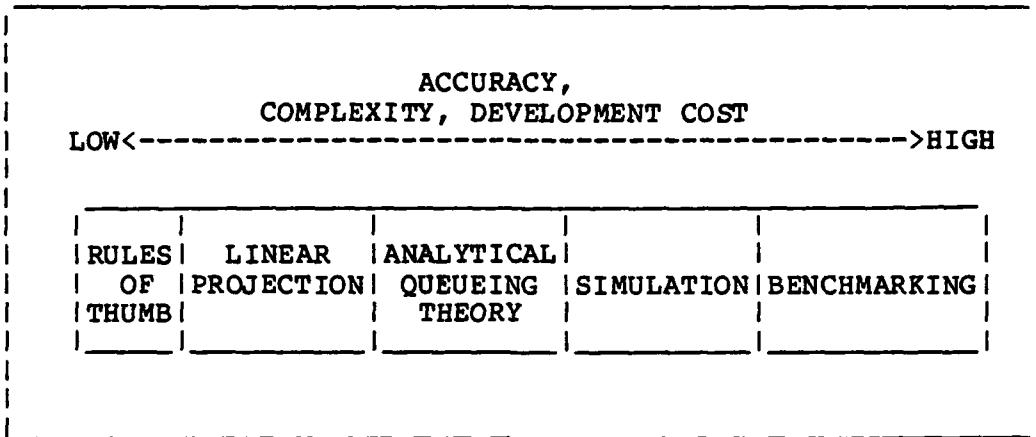


Figure 1-1 Spectrum of Computer Modeling Techniques (Ref 1:13)

These evaluation techniques involve increases in accuracy as one moves from the very simplistic rules of thumb to the very complex benchmarking techniques (see Figure 1-1). Unfortunately, the same increase in accuracy also involves an increase in cost. The computer engineer involved in CPE must then make tradeoffs as to the amount of accuracy he can afford. These tradeoffs are extremely prevalent when comparing the two most widely used types: simulation and analytical queueing models.

There are three reasons for the increasing popularity of analytical models over simulation models for computer system modeling (Refs 1, 4, 11, 21, 23, 26, and 29):

- Analytical models capture the most important features

Introduction

of actual systems, i.e. jobs moving from one queue to another waiting for service from independent devices within the system.

- The assumptions of the analysis are realistic. General device service time distributions, load-dependent devices service times, and multiple classes of jobs can be modeled.
- The algorithms that solve the equations of the model are available as highly efficient queueing network evaluation packages. Because of their efficiency and simplicity, these models are cheaper than simulations to develop and run.

1.1.3 Development of Analytical Models in CPE

The development of more flexible analytical modeling techniques for computer system modeling has paralleled the evolution of computer systems from single-programmed, batch systems to multi-programmed, combination interactive/batch systems.

1.1.3.1 Single Queue, Infinite Population Model

The earliest techniques modeled the entire computer system as a single resource with a single queue and an infinite population of jobs (see Figure 1-2). This model was used for the early batch computers, since the CPU was the dominate resource and only one job could be in the system at a time.

Introduction

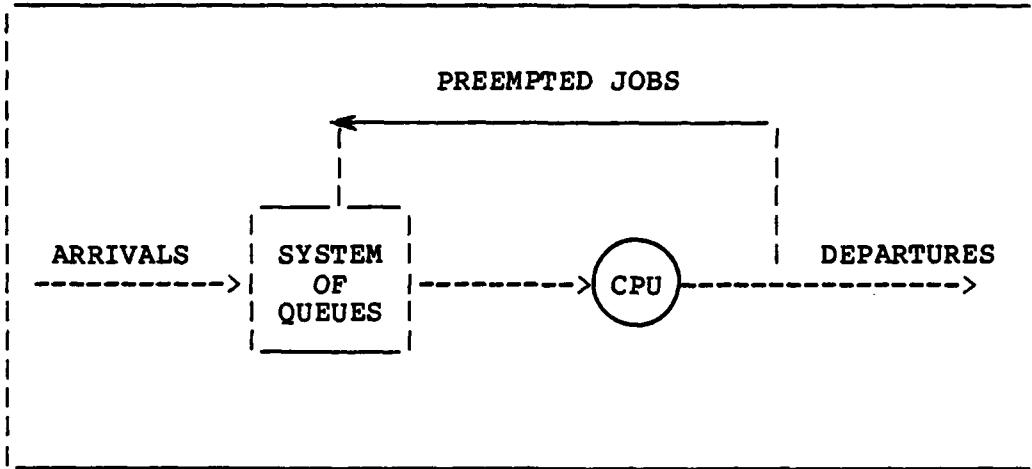


Figure 1-2 Single-resource Queueing Model (Ref 21:947).

1.1.3.2 Finite Population Model

With the advent of interactive systems, modeling the system as a single-resource queue with an infinite population of jobs became less realistic. Because there are a finite number of jobs circulating within an interactive system, the rate of arrival of new requests for service will tend to decrease as the queue length grows. The finite population model, also known as the machine interference model, was used to model this phenomena (see Figure 1-3). In the early interactive systems, only one complete job could be in memory at a time. The execution of the jobs, I/O activity, and the swapping activity were not overlapped. Therefore, program execution, I/O activity, and the swapping

Introduction

time were summed and used as the CPU service time (Ref 21:946-947).

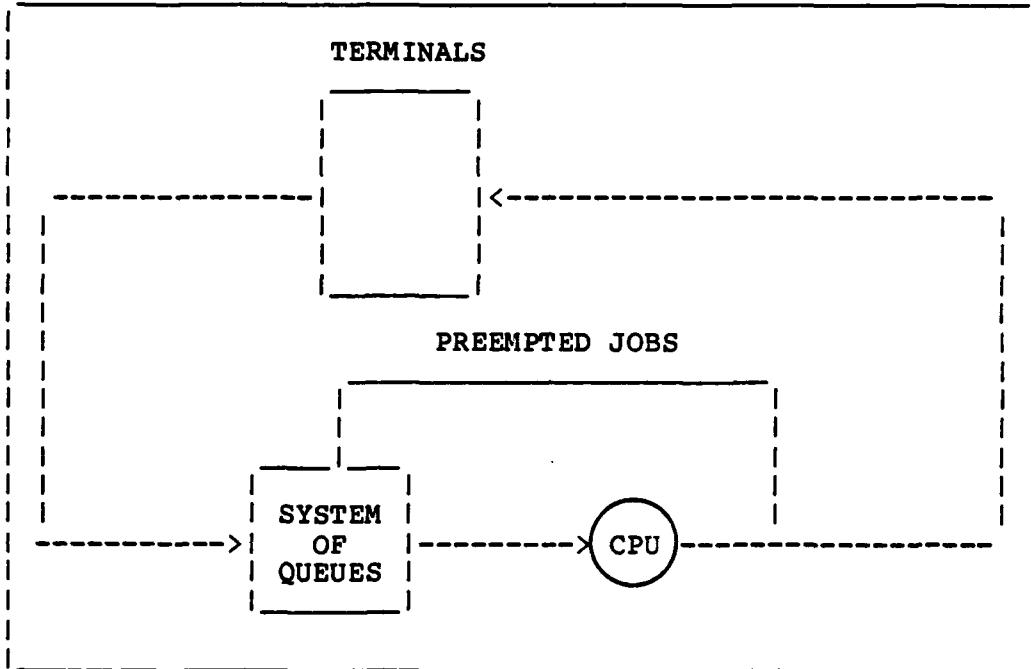


Figure 1-3 Finite Population Model (Ref 21:947).

1.1.3.3 Central Server Model

With the development of true multi-programming, i.e. more than one entire job within memory, the finite population models became less realistic. The most fundamental characteristic of jobs in a computer system is that they alternate between CPU execution and being blocked

Introduction

from further CPU execution waiting for access to secondary storage. With more than one job in memory, the I/O activity of one job could be overlapped with the execution of another job. A model which considers this type of behavior is shown in Figure 1-4. Note that this model still does not consider swapping times separately.

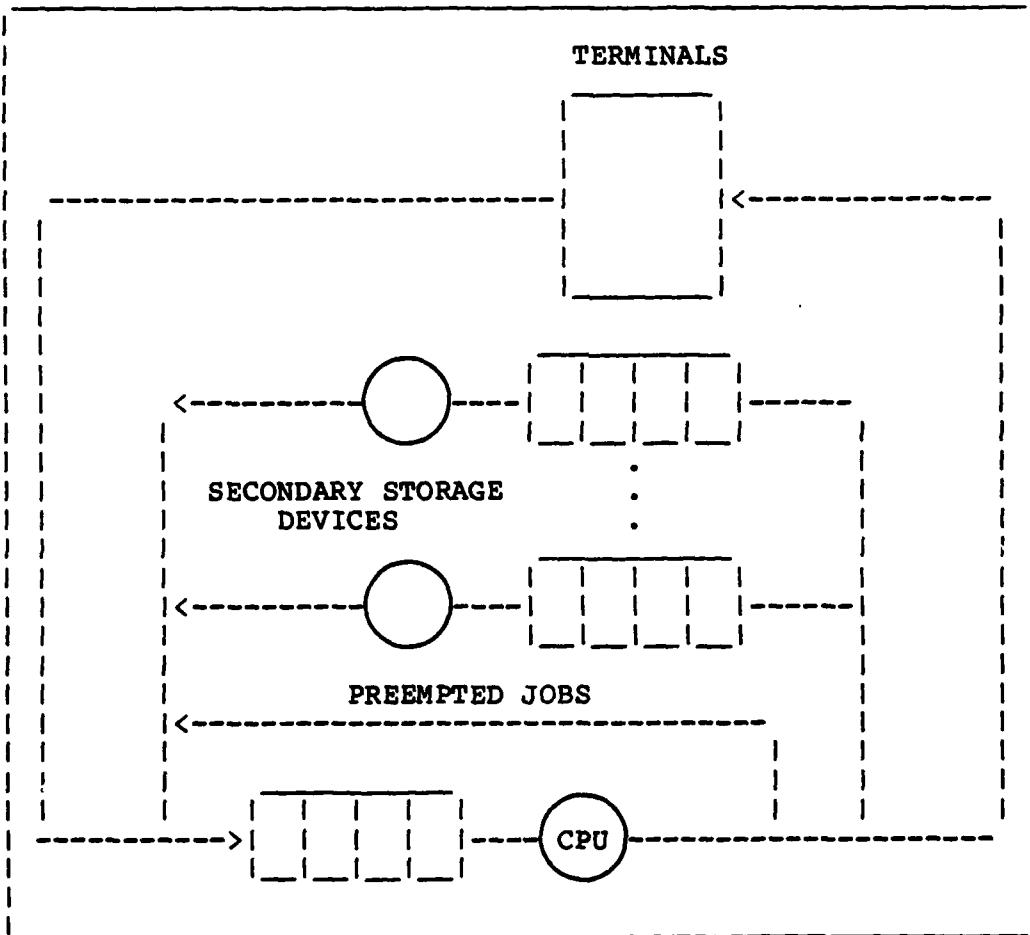


Figure 1-4 Central Server Model (Ref 3).

Introduction

1.1.3.4 Classical Swapping Model

As more and more computing systems used swapping as their memory management strategy, it became necessary to more accurately model the job swapping activities. The classical method to model job swapping explicitly was to add a swapping device node after the terminal interaction node (see Figure 1-5). Thus, each job is swapped out after each interaction with the user at the terminal (Ref 20). To improve the overall accuracy, two other improvements to the interactive model were developed (Ref 2:249)

1. Multiple classes of jobs with each class having its unique transition probabilities and node service times.
2. Non-exponential service times using the method of stages.

A more detailed account of these improvements and how they were integrated into one model is given in Chapter 3.

1.1.3.5 Chen's Swapping Model

Chen (Ref 5) proposed a new approach to modeling job-swapping behavior: that it be modeled by state-dependent transition probabilities in a closed queueing network with a

Introduction

single class of jobs. Since an exact solution to this problem does not currently exist, Chen developed an approximation algorithm that iteratively solves a closed queueing network. Chen's model is described in more detail in Chapter 3.

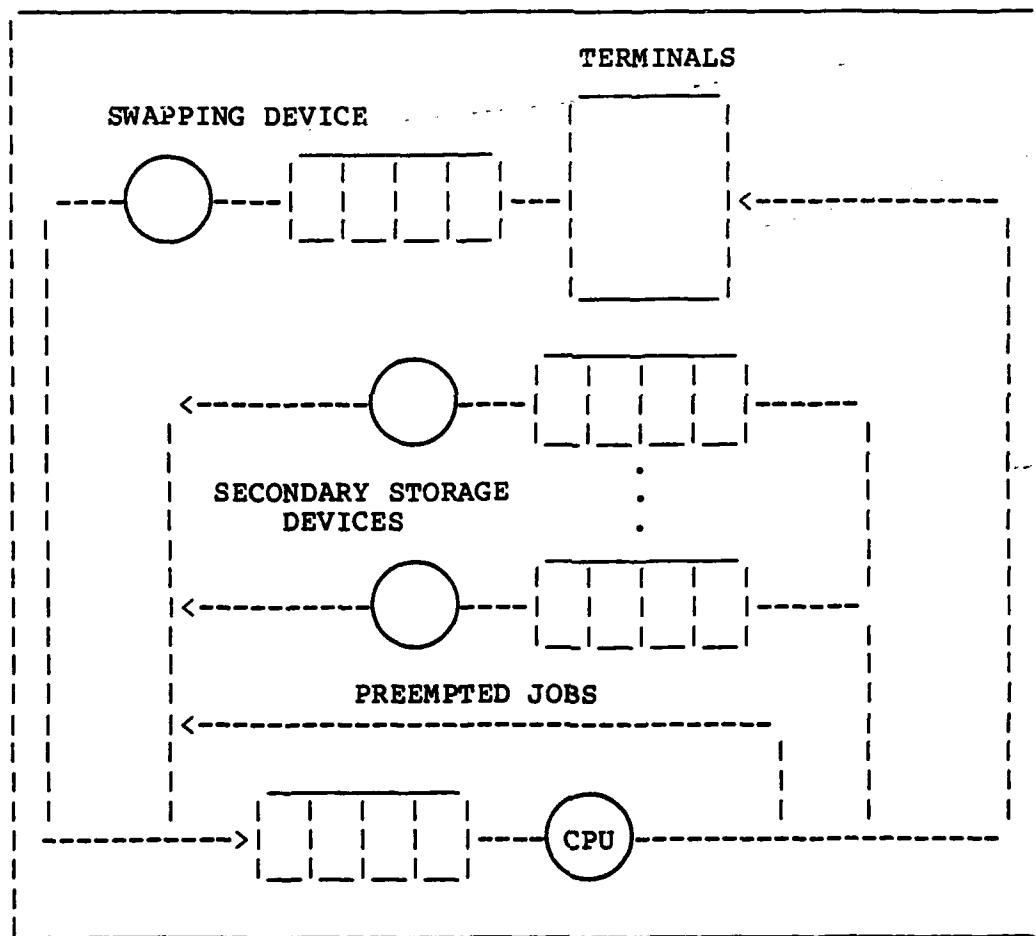


Figure 1-5 Classical Swapping Model (Ref 5).

Introduction

1.2 Problem Statement

1.2.1 Motivation for Research Objectives

The Avionics Laboratory at Wright-Patterson AFB is responsible for research, development, and validation of avionics systems for present and future aircraft in the Air Force inventory. This includes the development and validation of both the hardware (instrument displays, control consoles, etc.) and software (the computer programs that control the hardware) for navigational aids, weapons delivery systems, and electronic warfare systems. The laboratory is one of the most advanced facilities for the development and validation of avionics systems.

One of the primary test beds for avionics system development is the Digital Electronics Corporation's DECsystem-10 mainframe computer. The DECsystem-10 is a multi-programming, multi-processing, time-sharing computer with real-time processing capabilities. It provides high-level language support (FORTRAN, PASCAL, COBOL, BLISS, JOVIAL, and in the future ADA); assembly language support; document and manual preparation programs such as editors,

Introduction

text formatters, and spelling checkers; and graphics and plotting programs. Several mini-computers (PDP-11's) are directly interfaced to the DECsystem-10. This allows real-time communication between the DECsystem-10 and the PDP-11 which are used to control testing and simulation of current and experimental avionics systems.

Since the DECsystem-10 is such a vital Air Force resource in the research and development of state-of-the-art avionic systems, it is important that this resource is used as efficiently and effectively as possible. To insure the proper management of this resource, the system managers must have the proper tools to predict the impact of future workload changes, potential operating system changes, and reconfiguration of hardware resources. With these tools, the managers of the DECsystem-10 will gain a better understanding of the factors impacting computer performance and will be better able to make decisions to properly manage the DECsystem-10.

1.2.2 Background

In 1977 McKenzie developed a computer program to solve multi-class closed queueing networks with service centers having different service disciplines. This program was used

Introduction

to solve simple finite population and central server type queueing models of the DECsystem-10, the results of which helped the Avionics Lab to justify the purchase of additional memory. Both the results from McKenzie's limited validation and Saxton's entire thesis (Ref 24), indicated that more detailed models of the DECsystem-10 job-swapping behavior were needed (Refs 19:141 and 24:107).

1.2.3 Research Objectives

1.2.3.1 Primary Objective

The primary objective of this thesis effort was to improve the models of the DECsystem-10 by better modeling the job-swapping behavior. This will enable the system administrators to make more informed decisions concerning the present use of the computer system resources, as well as more accurate and responsive planning for future computer system acquisitions.

1.2.3.2 Specific Objectives

In order to meet the above primary objective, the following tasks were accomplished in this thesis effort:

1. The McKenzie program to compute performance measures for closed, multi-class queueing networks was brought

Introduction

up on the CDC computer. The program had to be typed in from a listing in the thesis and subsequently debugged in order to make it runnable.

2. The program was modified to include an approximation algorithm developed by Chen (Ref 5) to better model the DECsystem-10 swapping behavior. By combining the multi-class modeling capability of the McKenzie program with Chen's algorithm, we now have a new, more powerful tool to accurately model the performance of an interactive computer system with job-swapping.
3. The modified program was run using a workload consisting of interactive and batch jobs and the results used to answer the following questions:
 - Does Chen's swapping model improve the accuracy of the performance predictions when compared to the classical swapping model?
 - Does combining Chen's swapping model and multiple classes improve the accuracy of the performance predictions? Specifically, is it important to model an interactive computer system containing both interactive and batch jobs with a multi-class model versus using a single class model where
 - * The single class has the weighted average of the characteristics of the batch and interactive jobs in the multi-class model; or
 - * The single class has the characteristics of the largest class of jobs in the system?

1.2.4 Scope of the Thesis

The next chapter will discuss the objectives and basic functions of computer operating systems. This will be followed by a description of the DECsystem-10's operating system in enough detail to determine how it affects job

Introduction

swapping. Chapter 3 provides a detailed discussion of the state-of-the-art analytical queueing model with a product form solution developed by Basket, Chandy, Muntz, and Palacios. Chapter 3 concludes with a detailed description of Chen's swapping model and its underlying assumptions. Chapter 4 contains a description of McKenzie's computer implementation and how it was modified to include Chen's algorithm. Chapter 5 presents the results of modeling a hypothetical workload using the modified model to answer the above questions. Finally, Chapter 6 will present a summary of the conclusions as well as recommendations for future work in this area.

Chapter 2

Operating Systems and CPE

The primary method of improving the performance of an existing computer system without buying new and improved hardware is to modify the behavior of the program that controls the computer system. This program is called the operating system. Operating systems usually have built-in software parameters which may be set to allow the system administrator to implement various scheduling policies and priority schemes. Poorly chosen settings of these operating system parameters can cause inefficient use of system resources. Therefore, it is important that the system administrator has the tools to properly determine these operating system parameters. Before discussing this problem in detail and how it relates to the specific case dealt with in this paper, we will first review some of the basic objectives and functions of operating systems.

Operating Systems and CPE

2.1 Operating System Objectives

The operating system is a collection of system programs (algorithms) designed to meet the following objectives (Ref 6:2):

- Provide the programmers with an efficient environment for program development, debugging, and execution.
- Provide a range of problem-solving facilities (application programs).
- Provide all this at the lowest cost by sharing resources and information.

2.2 Resource Manager Functions

In order to meet the above objectives, the operating system must efficiently manage the system's resources. Therefore, its primary role is that of resource manager. It must accomplish the following (Ref 18:8):

1. Keep track of the resources.
2. Enforce policy that determines who gets what, when, and how much.
3. Allocate the resources.
4. Reclaim the resources.

Operating Systems and CPE

The operating system's primary role can be broken down by resource type: memory, processors, devices (disk drives, tape drives, printers, etc.), and information (programs and data such as editors, compilers, file directories and other software resources). Below are listed the resource management functions by type and typical names given to some of the routines that perform these functions (Ref 18:9-10):

- Memory Management Functions

1. Keep track of the resource (memory). What parts are in use and by whom? What parts are not in use (called free)?
2. If multiprogramming, i.e. more than one program in execution at one time, decide which job gets memory, when it gets it, and how much.
3. Allocate the resource (memory) when the jobs request it and the policy of 2 above allows it.
4. Reclaim the resource (memory) when the job no longer needs it or has been terminated.

- CPU Processor Management Functions

1. Keep track of the resource (processors and the status of jobs). The system program that does this has been called the traffic controller.
2. Decide who will have a chance to use the processor; the job scheduler chooses from all the jobs submitted to the system and decides which one will be allowed into the system, i.e. have resources assigned to it. If multiprogramming, decide which job gets the processor, when, and how much.
3. Allocate the resource (processor) to a job by setting up necessary hardware registers; this system program is often called the dispatcher.

Operating Systems and CPE

4. Reclaim the resource (processor) when the job relinquishes processor usage, exits from the system after job completion, or exceeds allowed amount of usage and aborts.

- Device Management Functions

1. Keep track of the resource (disk and tape drives, channels, control units); this is typically called the I/O traffic controller.
2. Decide what is an efficient way to allocate the resource (device). If it is to be shared, then decide who gets it, and how much he is to get; this is called I/O scheduling.
3. Allocate the resource (device) and initiate the I/O operation.
4. Reclaim the resource (device). In most cases when the I/O terminates, the device is released automatically, but the operating system must be informed of the new status of the device.

- Information Management Functions

1. Keep track of the resource (information), its location, use, status, etc. These collective facilities are often called the file system.
2. Decide who gets use of the resources, enforce protection requirements, and provide accessing routines.
3. Allocate the resource (information), e.g., open a file.
4. Deallocate the resource, e.g., close a file.

Operating Systems and CPE

2.3 The DECsystem-10 Operating System: TOPS-10 Monitor

The TOPS-10 Monitor performs the accounting, scheduling, resource allocation, and service routines necessary to operate in a multiprogramming, time-sharing environment. It both controls user jobs and provides services to them. The monitor gives the appearance of a single-user machine to all the users on the DECsystem-10 by rapidly switching control to each user. It manages all I/O operations, according to requests from user programs and from device interrupts. It attempts to allocate all system resources in such a way as to give the best overall system performance (Ref 8:INTRO-10).

2.3.1 Monitor Programs

The monitor consists of many separate and more or less independent programs which are called according to events which occur within the system. The system programs are divided into two types depending on whether the program executes synchronously or asynchronously. Synchronous

1. The clock interrupt is an AC line generated interrupt and therefore occurs every 1/60 of a second. This time interval has been given the name "jiffy" and the term occurs frequently in DECsystem-10 literature.

Operating Systems and CPE

system programs execute once between system clock interrupts¹ and complete their function before the next clock interrupt. After each clock interrupt, the monitor cycle is executed followed by the running of a user job selected by the Job Scheduler (Ref 8). The synchronous system programs are the

CLOCK1 The system program which acts as the "main" calling program for all other system programs. This program contains the code used by the Job Scheduler to make the actual resource assignments.

Job Scheduler The system program which selects the next job to run in the remainder of the jiffy and controls the allocation of system resources. This program is also known as SCHED.

Command Processor The system program which reads in a typed command and interprets it.

Context Switching Routine The system program which transfers control to the user job selected by the Job Scheduler.

Asynchronous system programs execute on an as needed basis by jobs within the system and may have program cycles lasting longer than one jiffy. The asynchronous system programs are described below

Job Swapper The system program called by the Job Scheduler to keep as many runnable jobs in memory as possible. Even though it is called each time during the execution of the Job Scheduler (a synchronous program), the Job Swapper is not synchronous since the swapping in and out of a job usually takes several jiffies to complete. The Job Swapper

Operating Systems and CPE

"remembers" what it was doing when called by the Job Scheduler. Each time the Job Swapper is called, it continues processing where it last terminated during the last jiffy.

UUO² Processor Routine

The system program which executes whenever a job requests I/O service such as disk reads or writes.

Core Management Routine

The system program which dynamically allocates memory as needed by other system programs or user jobs.

The system programs described above and how they interact with one another during a clock interrupt is shown in Figure 2-1. All dotted arrows represent asynchronous events which may or may not occur within a monitor cycle, depending on the state of the system (Ref 8).

2.3.2 Job States

The next state transition that a job can enter depends on the current state of the job (Ref 6:31-35). The job state is represented by many status variables maintained within the monitor system program and job status tables. A job can be in one of four states (Ref 8:5-1):

2. Unimplemented User Operations (UUO) are machine language operation codes which directly call the monitor to execute supervisor level tasks such as I/O requests to system peripherals.

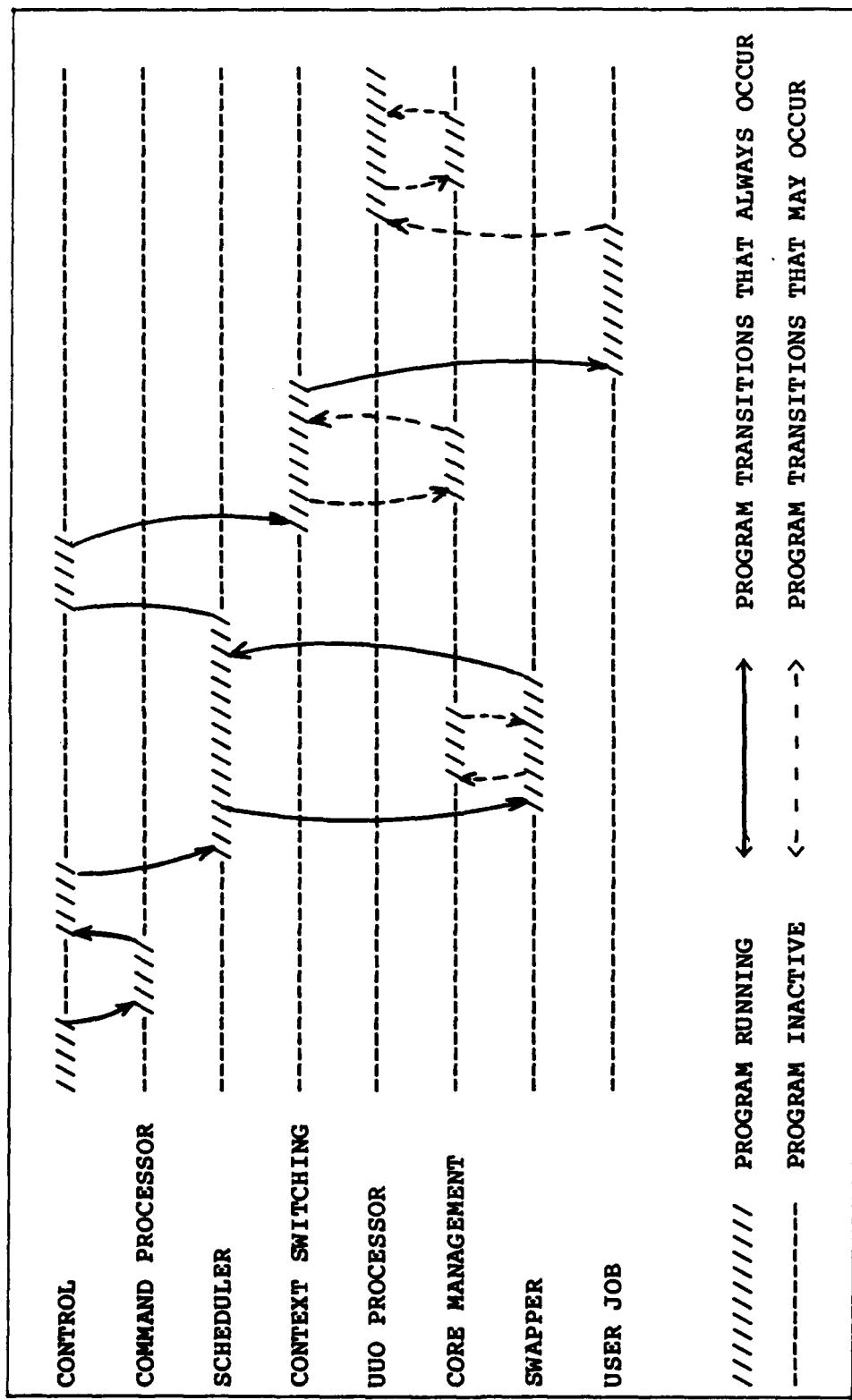


Figure 2-1 Monitor Cycle Programs

Operating Systems and CPE

1. The job is in a processor queue, has memory, and is running on a CPU.
2. The job is in a processor queue, ready to execute on a CPU. As defined by DECsystem-10 literature, a job in a processor queue is defined to be ready to execute if the job is not waiting for a shareable resource, even though the job may have no memory and must first be swapped in.
3. The job is in a short-term wait state. Jobs in this state were previously in the running state, but requested the use of a shareable resource. Jobs in the short-term wait state can not be selected to execute until they are assigned the shareable resource and are once again runnable.
4. The job is in a long-term wait state. Jobs in this state were previously in the running state, but then requested the use of a non-shareable resource. Non-shareable resources are distinguished from shareable resources by the length of time the job may have to wait to be assigned the resource. Non-shareable resources include such things as line printers or card readers, as well as less obvious non-shareable resource such as a response from a terminal.

The possible states of a DECsystem-10 are summarized in Figure 2-2. The states labeled HPQ 1-15, PQ1, and PQ2 are the processor queues and are discussed in the following section.

3. A shareable resource is some part of the system, either hardware or software, which can be used by only one job at a time, but is shared among different jobs over relatively short periods of time. Examples of shareable resources are I/O channels and disk drives used to satisfy disk I/O requests

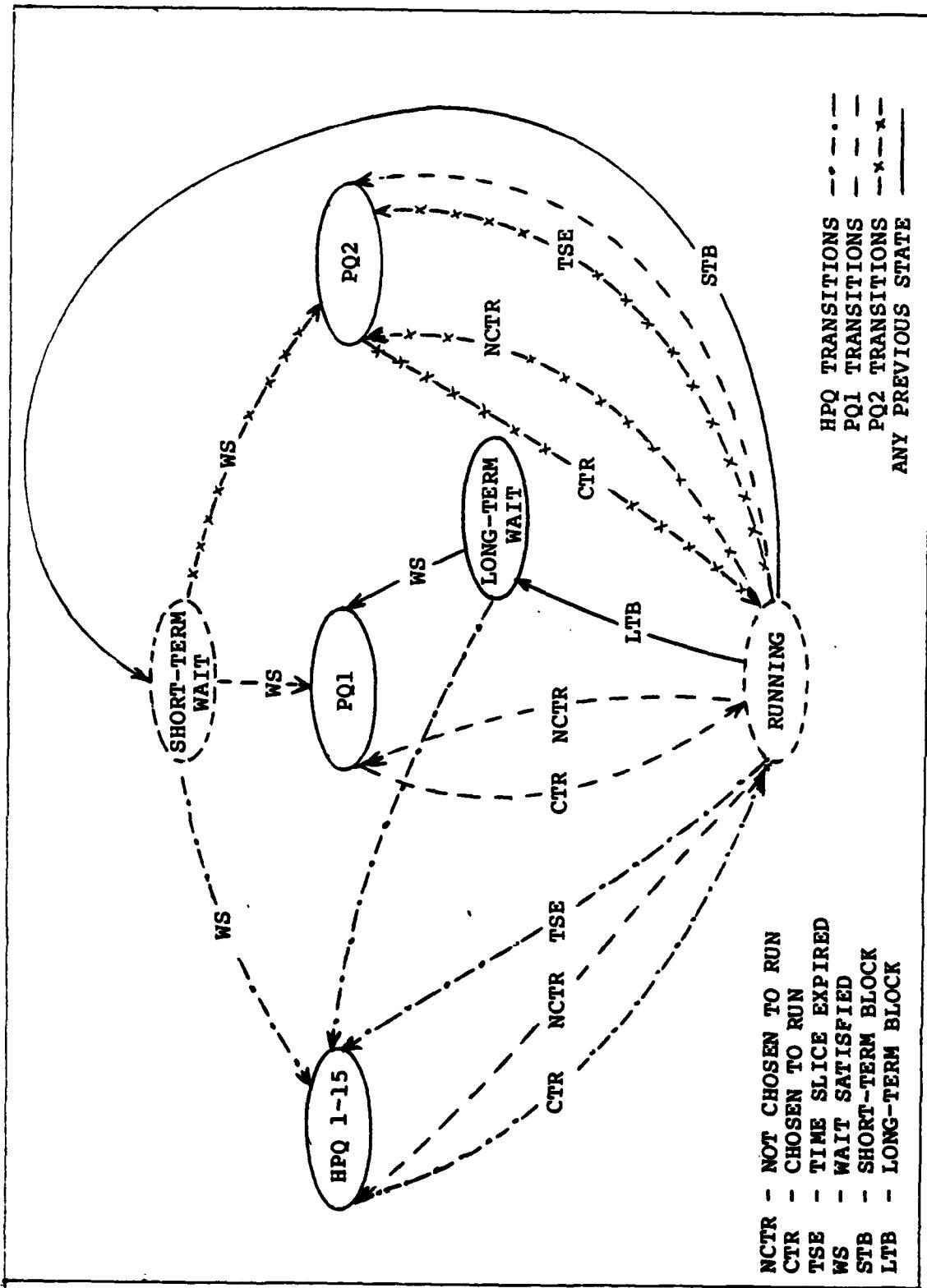


Figure 2-2 Job State Transitions

Operating Systems and CPE

2.3.3 Monitor Queueing Structure

Scheduling in the DECsystem-10 is based on the use of queues and wait state codes. The jobs in the system are maintained in a master set of queues, which are logically divided into long-term wait queues and processor queues.

2.3.3.1 In-core Versus Out-core Chains

Each queue in the long-term wait queues and processor queues are separated into two chains: one contains all the jobs in the queue that are in-core and the other contains all the jobs that are out-core (have no memory). This breakdown enables efficient scanning of the queues by the Job Scheduler and the Job Swapper. The scheduler only has to scan jobs in the in-core chains to find a runnable job with memory. The Job Swapper only has to scan jobs in the in-core chains to find a job to swap out to make room for a higher priority job or scan the out-core chains to find a job that needs to be swapped in (Ref 8:SCH-8).

2.3.3.2 Processor Queues

Jobs in the processor queues can either be running, ready to run (with or without memory), or in a short-term wait for a shareable resource. The processor queues are the

Operating Systems and CPE

high-priority queues (HPQ's), PQ1, and PQ2. Each are described below (Ref 28:2-1):

HPQ's	(Up to 15 levels, called HPQ1 through HPQ15) contain jobs that require real-time response, such as the line-printer spooler and the card-reader spooler system programs. These queues are scanned first by the Job Scheduler to find an executable job.
PQ1	Contains jobs that require fast response, such as time-sharing jobs. This queue is the next most often scanned processor queue.
PQ2	Contains jobs that require long-term computing, such as those that compile programs. Jobs in this queue are scanned least often by the Job Scheduler, since fast response is not necessary.

In addition to providing a priority structure for the Job Scheduler to scan for an executable job, it establishes a priority structure for the Job Swapper. The priority structure, though, is exactly reversed. The Job Swapper first scans PQ2, PQ1, and then the HPQ's to find a job it can swap out. The Job Swapper can not swap-out jobs performing disk I/O's and tries to avoid swapping out an executable job. Therefore, jobs in long-term wait states and other short-term wait states besides disk I/O's are the Job Swapper's primary target.

2.3.3.3 Long-term Wait Queues

The long-term wait queues hold jobs that are in a

Operating Systems and CPE

long-term wait state. The queues and their purpose are described below (Ref 28:2-3):

CMQ	Command Wait Queue. The user has typed a monitor command that cannot be executed until the command program is in memory, and the program is not in memory.
TIOWQ	Teletype I/O Wait Queue. Waiting for the user to type a response or waiting for the device to print output already sent to it.
JDCQ	DAEMON Wait Queue. The job is waiting for service by DAEMON. The DAEMON is a system program which runs as a user job and performs various functions such as recording accounting data or error logging required by other user jobs. It is, in effect, a non-resident portion of the monitor.
EWQ	Event Wait Queue. This queue encompasses many types of resource allocation waiting lines such as waiting for a magnetic tape controller, etc.

Jobs in a long-term wait queue are requeued to the rear of the PQ1 when their long-term wait is satisfied.

Describing the above as queues is actually a misnomer. Having a queue implies that some order exists in the queue, i.e. first-come-first serve (FCFS), last-in-first-out (LIFO), etc. In the long-term wait queues the order in which they leave the queue is not dependent on their order in the queue. Instead, the order in which they leave is totally dependent on the job characteristics. For example, even though user A may arrive before user B to the terminal

Operating Systems and CPE

I/O wait queue (TIOWQ), if user B responds first, he leaves the queue before user A.

The main purpose of having long-term wait queues is two fold:

- Remove jobs from the processor queues so that the Job Scheduler does not waste time scanning jobs that are not expected to become runnable for long periods of time.
- Provide a set of queues that the Job Swapper can scan first to find a job to swap-out to make room for other jobs.

The important thing to note here is that when there are more jobs than can fit in memory, jobs in the long-term wait queues will be swapped from the in-core chain to the out-core chain before jobs in the processor queues.

2.3.3.4 Processor Queue Time Slices

Other factors, besides which queue the job is in, affect the likelihood that a job will be swapped. These include how long it can be in memory without becoming eligible for swap-out and its position in the queue. These two factors are determined by the time slice assigned to a job.

When a job enters one of the processor queues, it is

Operating Systems and CPE

assigned a time slice. The time slice consists of two components: the in-core protect time (ICPT) and the quantum run time (QRT). The ICPT and QRT are actually counts. The ICPT is the number of times the Scheduler can try to execute the job before the job becomes eligible for swap-out. The QRT is the number of times the Scheduler can execute the job before it is requeued to the rear of a processor queue.

Each time the Job Scheduler scans a job to see if it is executable, the ICPT is decremented. Note that the Job Scheduler may not execute a job because it is in some short-term wait state, but the ICPT is still decremented. When this count reaches zero, the job becomes eligible for swap out. The ICPT provides a mechanism wherein the swapper is prevented from immediately swapping out a job which has just been swapped in, and in effect "locks" the job in memory until its ICPT is zero (Ref 8).

Each time the Job Scheduler chooses a job to execute, that job's QRT is decremented. When the QRT reaches zero, the job either requeues to a lower priority queue, e.g. PQ1 jobs requeue to the end of the PQ2 queue, or the job requeues to the rear of the queue it is in, e.g. PQ2 and HPQ jobs requeue to the rear of their respective queue. When treated in this way, the QRT limits the amount of CPU

Operating Systems and CPE

time the job receives in a particular queue, thus providing a fairness consideration in assigning CPU time. Also, since the Job Swapper usually scans the processor queues from back to front, jobs that have most recently expired their QRT and were requeued to the rear of the queue are more likely to be swapped out (Ref 8). This provides a fairness consideration in assigning the use of main memory.

PQ1 Time Slice

The PQ1 time slice is the amount of time that a job receives fast interactive response after being swapped in. The QRT of the PQ1 queue is usually much smaller than other processor queues in order to provide quick response to interactive jobs. The short QRT will be sufficient for interactive jobs before they become blocked to a long-term wait state, e.g. waiting for terminal response. The more CPU-intensive jobs will expire their QRT and be requeued to the lower priority PQ2 queue. There, they will be assigned a new, much larger QRT, but now have a lower priority for execution and a greater likelihood of being swapped out.

Operating Systems and CPE

PQ2 Time Slice

For PQ2 jobs, the parameters for ICPT and QRT control the bias of the scheduler for throughput versus response and for I/O versus CPU. Throughput versus response is controlled by increasing or decreasing the magnitude of both parameters. As the parameters are increased, jobs expire their time slices more slowly, swapping rate decreases, and less core is allocated for swapping. These effects improve throughput, but average response is correspondingly degraded because interactive jobs wait longer to swap in. When you decrease both parameters the effect is reversed. I/O versus CPU response is controlled by changing the ratio of ICPT to QRT. Increasing only QRT favors CPU-bound jobs. Increasing only ICPT favors I/O-bound jobs, while reducing it tends to favor CPU-bound jobs (Ref 28:5-3).

2.4 Chapter Summary

The following are the important aspects to remember about the TOPS-10 Monitor

- The Monitor executes every jiffy at which time the Job Scheduler selects a job to run the remainder of the jiffy.

Operating Systems and CPE

- Jobs in the system can be in one of four states
 1. Executing on the CPU.
 2. Ready to run but waiting for its turn to execute on the CPU.
 3. In a short-term wait state waiting for a shareable resource.
 4. In a long-term wait state waiting for a non-shareable resource.
- Jobs in the system wait in processor queues for the CPU or for shareable resources while jobs waiting for non-shareable resources wait in the long-term wait queues. Interactive jobs wait for terminal response in a long-term wait queue.
- The processor queues and the long-term wait queues are divided into in-core chains (jobs that have memory) and out-core chains (jobs without memory that will have to be swapped in).
- Jobs in the long-term wait queues will be swapped from the in-core chain to the out-core chain before jobs in the processor queues.
- The time slices assigned to jobs by the Job Scheduler are made up of two components: the incore protect time (ICPT) and the quantum run time (QRT).
 - * The ICPT provides a mechanism where in the swapper is prevented from immediately swapping out a job which has just been swapped in, and in effect "locks" the job in memory until its ICPT is zero.
 - * The QRT limits the amount of CPU time the job receives in a particular queue, thus providing a fairness consideration in assigning CPU time.
 - The short QRT in PQ1 will be sufficient for interactive jobs before they become blocked to a long-term wait state while more CPU-intensive jobs will expire their QRT and be requeued to PQ2 where they will be assigned a new, much larger QRT.

Operating Systems and CPE

- The longer QRT in PQ2 provides the extensive processing needs of jobs needing large amounts of CPU processing. The ratio of the PQ2 QRT and the ICPT determines the schedulers bias towards CPU-intensive jobs versus I/O intensive jobs.

Chapter 3

Analytical Modeling

3.1 Early Queueing Network Models

Jackson (Ref 14) and Gordon and Newell (Ref 10) give solutions for the case of networks of queues with the following assumptions

1. There are a finite number of nodes, M.
2. There are a fixed finite number of customers, N.
3. The manner in which customers visit the various resources is governed by a transition matrix $P = [p_{ij}]$, where p_{ij} is the probability that a customer departing from node i will next visit node j.
4. The service time distributions at nodes are exponentially distributed.
5. The service rate may be a function of the number of customers.
6. All customers are identical in that their routing probabilities and service times at the M nodes are the same.

For this class of models, a state of the model is given by the number of customers at each node. Thus the set of states is just

Analytical Modeling

$$(n_1, n_2, \dots, n_M) \mid \sum_{i=1}^M n_i = N$$

The equilibrium state probabilities are given by

$$P(n_1, n_2, \dots, n_M) = C \prod_{i=1}^M \frac{\lambda_i^{n_i}}{\prod_{j=1}^{n_i} u_i(j)}$$

where

λ_i The mean arrival rate to the i^{th} node.

$u_i(j)$ The instantaneous departure rate from the i^{th} resource when there are j customers queued at this node.

C The normalization constant is chosen so that all the equilibrium state probabilities sum to one.

The calculation of C can be very time consuming since the number of states is

$$\binom{M + N - 1}{N}$$

However, methods of calculating C have been found which increase only as MN^2 (Ref 3) allowing larger queueing networks to be solved.

Analytical Modeling

3.2 Multi-class Queueing Network Model

The earliest queueing models developed by Jackson and Gordon and Newell, and the computational algorithm derived by Buzen suffered from two limitations:

1. Only one class of customers was allowed in the network.
2. All the service time distributions were exponential.

Recently developed queueing models addressed many of the important aspects of applying queueing models to time-sharing computer systems such as multi-class jobs and non-exponential service time distributions. Still, no one unified model existed to describe all these aspects of time-sharing systems.

The model developed in reference 2 by Basket, Chandy, Muntz, and Palacios and described below accomplished this unification. It combined recent results on the networks of queues of several different service disciplines and a broad class of service time distributions with earlier results on networks of queues containing different classes of customers. This model thus more accurately describes a time-shared computer system.

Analytical Modeling

3.2.1 Variables in the Model

The model describes a system with an arbitrary but finite number (N) of service centers with an arbitrary but finite number (R) of job classes. The routes through the network of service centers for the R classes of jobs is determined by a probability transition matrix $P = [P_{i,r;j,s}]$, where $P_{i,r;j,s}$ is the probability of a class r job at service center i going to service center j as a class s job. The transition matrix defines a Markov chain assumed to be decomposable into m subchains, E_1, E_2, \dots, E_m .

Let n_{ir} be the number of jobs of class r at service center i in state S_j of the network model. Also let

$$M(S/E_j) = \sum_{(i,r) \in E_j} n_{ir}$$

be the total number of jobs within the subchain. Then

$$M(S) = \sum_{j=1}^m M(S/E_j)$$

Analytical Modeling

is the total number of jobs in the network. A system is closed when $M(S/E_j) = \text{constant}$, $1 < j < m$.

3.2.2 Open Network Outside Arrival Processes

In an open network the arrivals to node in a queueing network can be external from the network. The fixed probability of an outside arrival of a class r customer at service center i is q_{ir} . The probability of a class r job leaving the system from service center i is

$$1 - \sum_{\substack{1 < j < N \\ 1 < s < R}} p_{i,r;j,s}$$

The external arrival process can be state dependent and is of two general types

1. The arrival rate, $\lambda(M(s))$ is dependent on the total number of customers in the system, $M(S)$.
2. The arrival rate, $\lambda_j(M(S/E_j))$ is decomposed into m Poisson arrival streams corresponding to the m subchains described above. The m arrival rates are dependent on the number of customers in the subchain, $M(S/E_j)$.

Analytical Modeling

3.2.3 Service Time Distributions

Exponential, hyperexponential, and hypoexponential distributions are eligible service time distribution for this model. Also, any service time distribution that can be represented as a network of states can be used as a service time distribution. In this method of stages technique, u_{irl} represents the service rate of a job in service stage 1 at service center i who is in class r . There are s_{ir} service stages for a job in class r at node i . The probability of going to the next stage is a_{irl} , while the probability of completing service is b_{irl} . These concepts are illustrated in Figure 3-1.

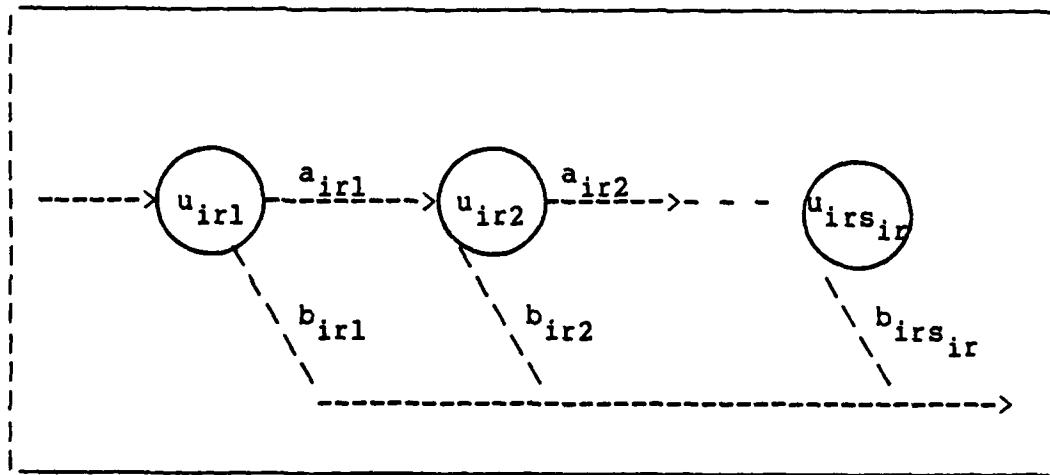


Figure 3-1 Method of Stages.

Analytical Modeling

3.2.4 State-dependent Service Rates

Three types of state-dependent service rates can be incorporated into the Basket, et. al. model

1. The service rate at a service center can depend on the total number of jobs at that service center. This form of state-dependent service rate is useful to model multiple, identical servers. If there are k_i identical servers at service center i , then by taking the service rate with one job present and multiplying it by the following function

$$x_i(n_i) = \begin{cases} n_i, & 1 \leq n_i \leq k_i \\ k_i, & n_i > k_i \end{cases}$$

we obtain the service rate for service center i .

2. The service rate of a job class can depend at a service center on the number n_{ir} of class r jobs at service center i .
3. The service rate of a service center can depend on the number of jobs at other service centers in the network.

Note that these various forms of state-dependent service rates can be mixed.

3.2.5 Service Center Types

Service centers can be one of four types:

1. The service discipline is first-come-first-served (FCFS).

Analytical Modeling

- All customers have the same service time distribution at this service center.
- The service time distribution must be a negative exponential.
- Only state-dependent service rate types 1 and 3 can be used.

This type of service center is most often used to model the I/O devices such as disks, drums, and tape drives in a computer system.

2. The service discipline is processor sharing (PS), i.e. when there are n_i customers in the service center, each is receiving service at a rate of $1/n_i$ times the service center's rate for its job class.

- Each class of customer can have a distinct service time distribution.
- All three service time distributions can be used.
- All three state-dependent service rate types can be used.

This type of service center is most often used to model a central processor that uses a round-robin scheduling algorithm. This is appropriate since round-robin scheduling approaches the processor sharing service time as the round-robin time quantum approaches zero.

3. The number of servers in the service center is greater than or equal to the maximum number of customers that can be at this center in a feasible state (this is the infinite server (IS) case).

- Each class of customer can have a distinct service time distribution.
- All three service time distributions may be used.
- All three state-dependent service rate types can be used.

Analytical Modeling

This type of service center is most often used to model the terminal users in a time-sharing computer system.

4. The service discipline is preemptive-resume last-come-first-served (LCFS)

- Each class of customer can have a distinct service time distribution.
- All three service time distributions can be used.
- All three state-dependent service rate types can be used.

This type of service center is most often used to model a central processor in which jobs can preempt other jobs using the processor.

3.2.6 States of the Model

The state of the model is represented by a vector (x_1, x_2, \dots, x_N) where x_i represents the state of service center i . The representation of the state of the service center is dependent on the type of service center.

3.2.6.1 Type 1 Service Center

If service center i is of type 1, then $x_i = (x_{i1}, x_{i2}, \dots, x_{ik})$, where $k = n_i$ is the number of customers at center i and x_{ij} ($1 < j < n_i$, $1 < x_{ij} < R$) is the class of customer who is j th in FCFS order. The first customer is

Analytical Modeling

served while the remainder are waiting for service. This type of service center state space is very large since one must not only account for the number of customers of each type, but the order of the queue must be properly represented.

3.2.6.2 Type 2 and Type 3 Service Centers

If service center i is of type 2 or 3, then $x_i = (v_{i1}, v_{i2}, \dots, v_{iR})$, where v_{ir} is a vector $(m_{1r}, m_{2r}, \dots, m_{kr})$ where $k = s_{ir}$. The l^{th} component of v_{ir} is the number of customers of class r in center i and in the l^{th} stage of service. The number of stages for a class r customer at service center i is s_{ir} . Note here that a state is distinguished from another state by the number of customers, the class of the customer, and the stage of service of the customer, but not the order of the customers at the service center since there is no waiting line for these service disciplines.

3.2.6.3 Type 4 Service Centers

If the service center is of type 4, then $x_i = ((r_1, m_1), (r_2, m_2), \dots, (r_n, m_n))$ where the ordered pair (r_j, m_j) describes the j^{th} customer in LCFS order,

Analytical Modeling

i.e. r_j is the class of the customer and m_j is the stage of service of the customer. Note again that like the type 1 service centers, a queue exists and one must take into account the order of customers in the queue if there is more than one class.

3.2.7 Calculating Equilibrium State Probabilities

3.2.7.1 Balance Equations

The balance equation technique for the solution of equilibrium state probabilities is based on the concept that the rate of customers transitioning into a state is equal to the rate of customers transitioning out of the state, or more formally, for all states, S_j ,

$$\sum_{\substack{\text{all states} \\ S_j}} P(S_j) [\text{rate of flow } S_j \rightarrow S_i] = P(S_i) [\text{rate of flow out of } S_i]$$

These are the global balance equations of the queueing network. The balance equations establish a set of linear equations which can then be solved for the equilibrium state probabilities.

Analytical Modeling

A second type of balance equation for a queueing network exists, that of independent (local) balance. This concept equates the rate of flow into a state by a customer entering a stage of service to the flow out of that state due to the customer leaving that stage of service. A customer can be associated with a stage of service in the following ways

1. If the customer is in service at a service center (always the case in type 2 or 3 service centers), then he is in one of the stages of his service time distribution at that service center.
2. If the customer is queued at a service center (only possible for type 1 or 4 service centers), then he is in the stage of his service time distribution he will enter when he is next served. For FCFS this is stage 1; for LCFS this is the stage of service when last preempted.

3.2.7.2 Product Form Solution

Before presenting the solution to this type of queueing network, we must first define a few additional terms.

Additional Notation

Each of the subchains defined above has associated with it a set of linear equations of the form

Analytical Modeling

$$\sum_{(i,r) \in E_k} e_{ir} p_{i,r;j,s} + q_{js} = e_{js}, \quad (j,s) \in E_k$$

where q_{js} is the rate of exogenous arrivals of class s customers to service center j . If $q_{js}=0 \quad (j,s) \in E_k$, then the network is closed with respect to E_k . In the case of closed networks, the above linear equations do not provide a unique solution. By setting the value of one of the e_{ij} a unique solution can be obtained. For convenience, one of the e_{ir} is set to 1, which allows the rest of the e_{ir} to be interpreted as relative arrival rates of class r customers to service center i (relative in that the arrival rates e_{ir} times the rate of the e_{ir} that was set to 1). Note that the network may be closed with respect to less than m of the subchains E_k .

One more term that appears in the product form solution needs to be defined. This is the probability that the r^{th} class at service center i is in the l^{th} stage of service, denoted as

$$A_{irl} = \prod_{j=1}^l a_{irj}$$

where a_{irj} is defined as in Figure 3-1.

Analytical Modeling

The Theorem

For a network of service stations which is open, closed, or mixed in which each service center is of type 1, 2, 3, or 4, the equilibrium state probabilities are given by

$$P(S = x_1, x_2, \dots, x_N) = C d(S) f_1(x_1) f_2(x_2) \dots f_N(x_N)$$

where C is a normalizing constant chosen to make the equilibrium state probabilities sum to 1, $d(S)$ is a function of the number of customers in the system, and each f_i is a function that depends on the type of service center i .

If service center i is of type 1, then

$$f_i(x_i) = (1/u_i)^{n_i} \prod_{j=1}^{n_i} [e_i x_{ij}]$$

If service center i is of type 2, then

$$f_i(x_i) = n_i! \prod_{r=1}^R \prod_{l=1}^{s_{ir}} [e_{ir} A_{irl}/u_{irl}]^{m_{irl}} (1/m_{irl}!)$$

Analytical Modeling

If service center i is of type 3, then

$$f_i(x_i) = \prod_{r=1}^R \prod_{l=1}^{s_{ir}} [e_{irl} A_{irl} / u_{irl}]^{m_{irl}(1/m_{irl})}$$

If service center i is of type 4, then

$$f_i(x_i) = \prod_{j=1}^{n_i} [e_{ir_j} A_{ir_j} m_j (1/u_{ir_j} m_j)]$$

If the arrivals to the system depend on the total number of jobs in the system, $M(S)$, and the arrivals are of class r and for center i according to fixed probabilities p_{ir} , then

$$d(S) = \prod_{i=0}^{M(S)-1} \lambda(i).$$

If we have the second type of state-dependent arrival process, then

$$d(S) = \prod_{j=1}^m \prod_{i=0}^{M(S/E_j)-1} \lambda(i).$$

If the network is closed, $d(S) = 1$.

Analytical Modeling

3.2.8 Example Problem

To illustrate the concepts defined above, a very simple example will be formulated. First, the global and independent balance equations for the problem will be defined using the "rate-in = rate-out" concept. Then the

Table 3-1 Transition Matrix and Service Rates.

Probability Transition Matrices by Job Class

Job Class 1		Job Class 2	
0.6	0.4	0.8	0.2
1.0	0.0	1.0	0.0

Service Rates by Job Class

Node	Job Class 1	Job Class 2	
1	1.0	1.0	(processor sharing)
2	2.0	1.0	(infinite server)

$$N_1 = 2 \quad N_2 = 2 \quad s_{ir} = 1 \quad R = 2$$

Analytical Modeling

equilibrium state probabilities will be solved for using the product form solution. These solutions will be verified by substituting the values in some of the balance equations to demonstrate that they do indeed solve the balance equations.

The example will consist of a closed queueing network with two service centers. There will be two classes of jobs with two jobs from each class in the network. Service center 1 will use the processor-sharing service discipline, while service center 2 will use the infinite server service discipline. This network is a very simple model of an interactive time-sharing computer system where service center 1 represents the computer and service center 2 represents the terminal users. The values of the probability transition matrix and service rates are shown in Table 3-1 along with a summary of the problem statement.

3.2.8.1 The Balance Equations

The state of the system can be represented as an ordered 4-tuple $(n_{11}, n_{21}, n_{12}, n_{22})$ where the first subscript represents the service center and the second represents the class. Therefore, n_{11} is the number of jobs at service center 1 from job class 1.

Analytical Modeling

Given the transition matrix and service rates in Table 3-1, the state transition diagram is as shown in Figure 3-2. Using the diagram in Figure 3-2 the global and local balance equations can be derived. The global balance equations of this example are shown in Figure 3-3. The local balance equations are shown in Figure 3-4.

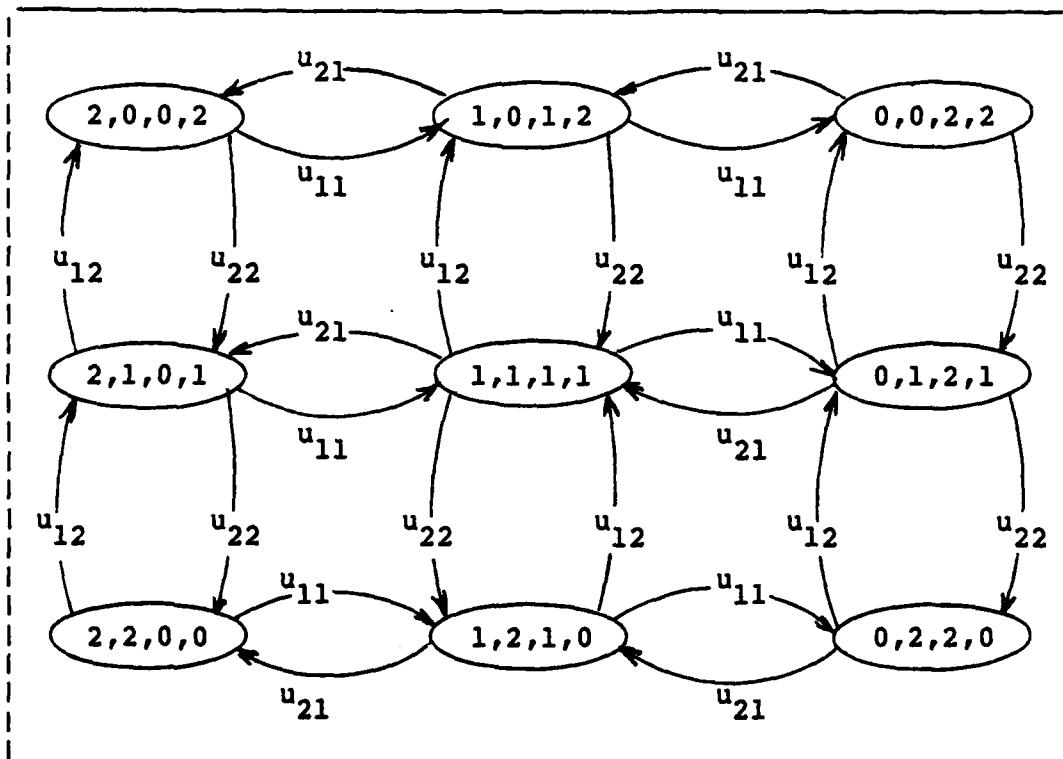


Figure 3-2 State Transition Diagram.

1. $P(2,0,0,2)[u_{22}(2)(1) + u_{11}(2/(0+2))(.4+.6)] = P(2,1,0,1) u_{12}(1/(1+2)) (.2) +$
 $P(1,0,1,2)u_{21}(1)(1) + P(2,0,0,2)u_{11}(2/(0+2)) (.6)$

2. $P(1,0,1,2)[u_{21}(1)(1) + u_{11}(1/(1+0)) (.4+.6) + u_{22}(2)(1)] =$
 $P(2,0,0,2)u_{11}(2/(0+2)) (.4) + P(1,1,1,1)u_{12}(1/(1+1)) + P(0,0,2,2) u_{21}(2)(1) +$
 $P(1,0,1,2)u_{11}(1/(1+0)) (.6)$

3. $P(0,0,2,2)[u_{21}(2)(1)+u_{22}(2)(1)] = P(1,0,1,2)u_{11}(1/(0+1)) (.4) +$
 $P(0,1,2,1)u_{12}(1/(0+1)) (.2)$

4. $P(2,1,0,1)[u_{12}(1/(1+2)) (.2+.8) + u_{11}(2/(2+1)) (.4+.6) + u_{22}(1)(1)(1)] =$
 $P(2,0,0,2)u_{22}(1/(0+1))(1) + P(1,1,1,1)u_{21}(1)(1) + P(2,2,0,0)u_{12}(1/(2+1)) (.2) +$
 $P(2,1,0,1)u_{11}(2/(2+1)) (.6) + P(2,1,0,1)u_{12}(1/(2+1)) (.8)$

5. $P(1,1,1,1)[u_{11}(1/(1+1)) (.4+.6) + u_{12}(1/(1+1)) (.2+.8) + u_{21}(1)(1)] =$
 $P(1,0,1,2)u_{22}(2)(1) + P(0,1,2,1)u_{21}(2)(1) + P(1,2,1,0)u_{12}(2/(1+2)) (.2) +$
 $P(2,1,0,1)u_{11}(2/(2+1)) (.4) + P(1,1,1,1)u_{11}(1/(1+1)) (.6) +$
 $P(1,1,1,1)u_{12}(1/(1+1)) (.8)$

Figure 3-3 Global Balance Equations

6. $P(0,1,2,1)[u_{12}(1/(0+1))(.2+.8) + u_{21}(2)(1) + u_{22}(1)(1)] = P(0,0,2,2)u_{22}(2)(1) +$
 $P(0,2,2,0)u_{12}(2/(0+2))(.2) + P(1,1,1,1)u_{11}(1/(1+1))(.6) +$
 $P(1,1,1,1)u_{12}(1/(1+1))(.8)$

7. $P(2,2,0,0)[u_{11}(2/(2+2))(.4+.6) + u_{12}(2/(2+2))(.2+.8)] = P(2,1,0,1)u_{22}(1)(1) +$
 $P(1,2,1,0)u_{21}(1)(1) + P(2,2,0,0)u_{11}(2/(2+2))(.6) + P(2,2,0,0)u_{12}(2/(2+2))(.8)$

8. $P(1,2,1,0)[u_{11}(1/(1+2))(.4+.6) + u_{12}(.2+.8) + u_{21}(1)(1)] = P(1,1,1,1)u_{22}(1)(1) +$
 $P(0,2,2,0)u_{21}(2)(1) + P(2,2,0,0)u_{11}(2/(2+2))(.4) + P(1,2,1,0)u_{11}(1/(1+2))(.6) +$
 $P(1,2,1,0)u_{12}(2/(1+2))(.8)$

9. $P(0,2,2,0)[u_{12}(2/(0+2))(.2+.8) + u_{21}(2)(1)] = P(0,1,2,1)u_{22}(1)(1) +$
 $P(1,2,1,0)u_{11}(1/(1+2))(.4) + P(0,2,2,0)u_{12}(2/(2+2))(.8)$

Figure 3-3 Global Balance Equations (cont.)

1. $P(2,0,0,2)[u_{11}(2/(0+2))(.4 + .6)] = P(2,0,0,2)u_{11}(2/(0+2))(.6) + P(1,0,1,2)u_{21}(1)(1)$
2. $P(2,0,0,2)[u_{22}(2)(1)] = P(2,1,0,1)u_{12}(1/(2+1))(.2)$
3. $P(1,0,1,2)[u_{11}(1/(1+0))(.4 + .6)] = P(1,0,1,2)u_{11}(1/(1+0))(.6) + P(0,0,2,2)u_{21}(2)(1)$
4. $P((1,0,1,2)[u_{21}(1)(1)] = P(2,0,0,2)u_{11}(2/(0+2))(.4)$
5. $P(1,0,1,2)[u_{22}(2)(1)] = P(1,1,1,1)u_{12}(1/(1+1))(1)$
6. $P(0,0,2,2)[u_{21}(2)(1)] = P(1,0,1,2)u_{11}(1/(0+1))(.4)$
7. $P(0,0,2,2)[u_{22}(2)] = P(0,1,2,1)u_{12}(1/(0+1))(.2)$
8. $P(2,1,0,1)[u_{11}(2/(2+1))(.4+.6)] = P(2,1,0,1)u_{11}(2/(2+1))(.6) + P(1,1,1,1)u_{21}(1)(1)$

Figure 3-4 Independent (Local) Balance Equations

9. $P(2,1,0,1)[u_{12}(1/(2+1))(.2+.8)] = P(2,1,0,1)u_{12}(1/(2+1))(.8) +$
 $P(2,0,0,2)u_{22}(1/(0+1))(1)$

10. $P(2,1,0,1)[u_{22}(1)(1)] = P(2,2,0,0)u_{12}(1/(2+1))(.2)$

11. $P(1,1,1,1)[u_{11}(1/(1+1))(.4+.6)] = P(1,1,1,1)u_{11}(1/(1+1))(.6) +$
 $P(0,1,2,1)u_{21}(2)(1)$

12. $P(1,1,1,1)[u_{12}(1/(1+1))(.2+.8)] = P(1,1,1,1)u_{12}(1/(1+1))(.8) +$
 $P(1,0,1,2)u_{22}(2)(1)$

13. $P(1,1,1,1)[u_{21}(1)(1)] = P(2,1,0,1)u_{11}(2/(2+1))(.4)$

14. $P(1,1,1,1)[u_{22}(1)(1)] = P(1,2,1,0)u_{12}(2/(1+2))(.2)$

15. $P(0,1,2,1)[u_{12}(1/(0+1))(.2+.8)] = P(0,1,2,1)u_{12}(1/(0+1))(.8) +$
 $P(0,0,2,2)u_{22}(2)(1)$

16. $P(0,1,2,1)[u_{21}(2)(1)] = P(1,1,1,1)u_{11}(1/(1+1))(.4)$

Figure 3-4 Independent (Local) Balance Equations (cont.)

$P(0,1,2,1)[u_{22}(1)(1)] = P(0,2,2,0)u_{12}(2/(0+2))(.2)$
 17. $P(2,2,0,0)[u_{11}(2/(2+2))(.4+.6)] = P(2,2,0,0)u_{11}(2/(2+2))(.6) +$
 $P(1,2,1,0)u_{21}(1)(1)$

18. $P(2,2,0,0)[u_{12}(2/(2+2))(.2+.8)] = P(2,2,0,0)u_{12}(2/(2+2))(.8) +$
 $P(2,1,0,1)u_{22}(1)(1)$

19. $P(1,2,1,0)[u_{11}(1/(1+2))(.4+.6)] = P(1,2,1,0)u_{11}(1/(1+2))(.6) +$
 $P(0,2,2,0)u_{21}(2)(1)$

20. $P(1,2,1,0)[u_{11}(2/(1+2))(.2+.8)] = P(1,2,1,0)u_{12}(2/(1+2))(.8) +$
 $P(1,1,1,1)u_{22}(1)(1)$

21. $P(1,2,1,0)[u_{21}(1)(1)] = P(2,2,0,0)u_{11}(2/(2+2))(.4)$

22. $P(0,2,2,0)[u_{12}(2/(0+2))(.2+.8)] = P(0,2,2,0)u_{12}(2/(0+2))(.8) +$
 $P(0,1,2,1)u_{22}(1)(1)$

23. $P(0,2,2,0)[u_{21}(2)(2)] = P(1,2,1,0)u_{11}(1/(1+2))(.4)$

Figure 3-4 Independent (Local) Balance Equations (cont.)

Analytical Modeling

3.2.8.2 The Product Form Solution

Since there are no stages of service in either service center, the function $f_1(x_1)$ and $f_2(x_2)$ can be simplified to the following

$$f_1(x_1) = n_1! \prod_{r=1}^2 [e_{1r}/u_{1r}]^{m_{1r}} (1/m_{1r}!)$$

$$f_2(x_2) = \prod_{r=1}^2 [e_{2r}/u_{2r}]^{m_{2r}} (1/m_{2r}!)$$

Solving for the e_{ir} above for this example, we get the following linear equations for class 1 jobs

$$e_{11} = .6e_{11} + e_{21}$$

$$.4e_{11} = e_{21}$$

and for class 2 jobs

$$e_{12} = .8e_{12} + e_{22}$$

$$.2e_{12} = e_{22}$$

which yield the following solutions for the e_{ir} 's

Analytical Modeling

$$e_{11} = 1.0$$

$$e_{21} = 0.4$$

$$e_{12} = 1.0$$

$$e_{22} = 0.2$$

Table 3-2 Example Calculations

State	$f_1(x_1)$	$f_2(x_2)$	P(state)
(2,2,0,0)	3/2	1	.1.5C = .670
(1,2,1,0)	3/4	.4	.3C = .134
(2,1,0,1)	3/2	.2	.3C = .0356
(1,1,1,1)	1	.08	.08C = .0356
(0,1,2,1)	1/2	.016	.008C = .00356
(1,0,1,2)	1	.008	.008C = .00356
(0,0,2,2)	1	.0016	.0016C = .000715
(0,2,2,0)	1/4	.08	.02C = .00894
(2,0,0,2)	1	.02	<u>.02C = .00894</u>
			$2.2376C = 1 \Rightarrow C = .4469$

The results of the computation are shown in Table 3-2.
The solutions can be verified by substituting them into some

Analytical Modeling

of the balance equations. For example, substituting the equilibrium probabilities into the 24th independent balance equation, we get

$$(.00894)(1)(2)(1) = (.134)(1)(1/3)(.4)$$
$$.01787 = .01787$$

and for the 17th balance equations

$$(.003575)(1)(1)(1) = (.00894)(2)(1)(.2)$$
$$.003575 = .003575$$

Therefore the product form solution of the above theorem provides solutions to the balance equations of this problem.

3.3 Chen's Swapping Model

The classical swapping model described in Chapter 1, along with the inclusion of multi-class jobs and non-exponential service times, provides a very flexible model of a computer system and its job swapping behavior. But the model still is not completely realistic; job

Analytical Modeling

swapping behavior is more complex and should depend on the main memory size, the number of jobs competing for memory, and the job sizes.

Chen (Ref 5) proposed a new approach: that job swapping behavior be modeled by state-dependent routing probabilities in a closed queueing network with a single class of jobs. The probability that a program needs to be swapped in (or out) is expressed as a function of the system state, the individual job characteristics, and main memory size. Unfortunately, this type of queueing model has no known exact solution. Therefore, Chen developed a successive-approximation algorithm that iteratively solves a closed queueing network. When two successive iterations yield results that are within specified tolerances of each other, the swapping model is solved. Chen's model and algorithm are described below.

3.3.1 Variables in the Model

First we will consider a simple model of an interactive computer system described by Chen (Ref 5) and shown in Figure 3-5. This system consists of one CPU, one disk, one swapping drum, and a set of terminals. Queues exist for the CPU, swapping drum, and disk, but no queue is necessary for

Analytical Modeling

the terminal node since there is a "server" (terminal user) at each active terminal.

The variables in the model are

- Number of jobs in the system (N).
- Main memory size (user area) (M).
- Average job size (J).
- Average CPU service time ($1/u_1$).
- Average disk I/O time ($1/u_2$).
- Average time to swap a job in and out of memory ($1/u_3$).
- Average user think time ($1/u_4$).
- Average CPU time needed per interaction (T_{CPU}).
- Average number of disk I/O requests per interaction (N_{DIO}).
- Average job size (J).

These variables above are used to calculate the probability transition matrix $P = P_{ij}$ where P_{ij} is the probability a job being serviced at node i will request the service of the node j. In terms of the context of the model, the meaning of some of the important P_{ij} 's are given below

P_{43}	The probability that the job is not in main memory and needs to be swapped in.
P_{41}	The probability that the job is already in main memory and does not need to be swapped in.

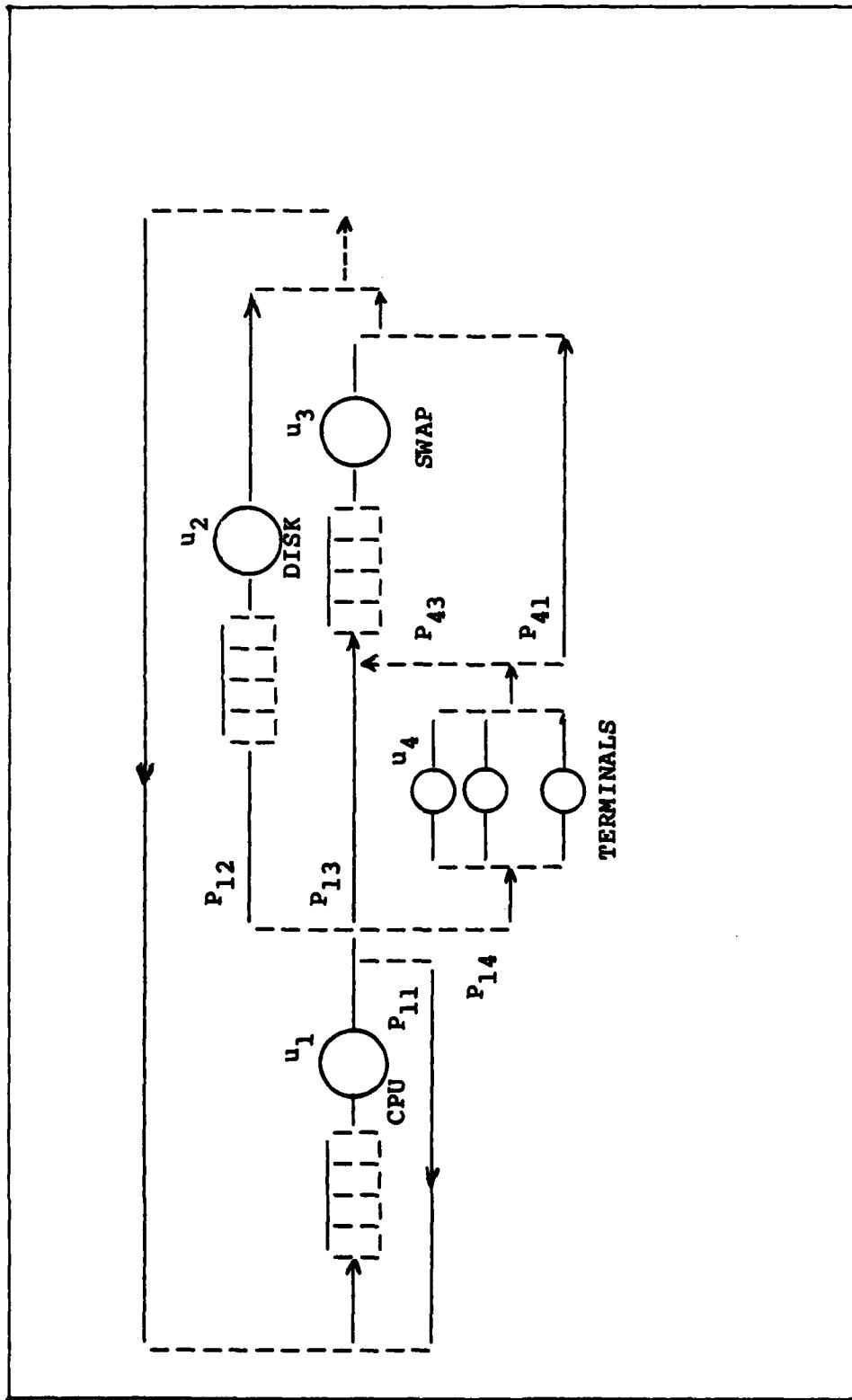


Figure 3-5 Chen's Swapping Model

Analytical Modeling

P_{11} The probability that the job's time quantum has expired

P_{12} The probability that the job needs information on a disk file or needs to write its results to a disk file.

P_{13} The probability that the main memory size is not sufficient and the job must be swapped out.

The state-dependent probabilities in Chen's algorithm are P_{11} , P_{43} , P_{41} , and P_{13} . Note that P_{41} and P_{13} are always zero in the simpler classical swapping model contained in Chapter 1. A general form of the probability transition matrix in Chen's model is shown in Table 3-3.

3.3.1.1 Derivation of P_{14} and P_{12}

Consider a typical sequence of events where 'i' represents the job leaving the CPU for an interaction at the terminal node and 'c' represents the job requesting another CPU service of $1/u_1$ seconds (on the average). A typical sequence would then look like

cc...cicc...cicc...

The number of CPU requests before an interaction would then be geometrically distributed and the average number of CPU service completions per interaction, N_{CPU} , would be

Analytical Modeling

$$N_{CPU} = \sum_{k=1}^{\infty} k P_{14} (1 - P_{14})^{k-1} = 1/P_{14}$$

(1)

**Table 3-3 Probability Transition Matrix
for Chen's Swapping Model**

P_{11}	P_{12}	P_{13}	P_{14}
1.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0
P_{41}	0.0	P_{43}	0.0

The average CPU time needed per interaction is equal to the product of the average number of times the job requests CPU service times the service rate or

$$T_{CPU} = 1/u_1 N_{CPU} = (1/u_1)(1/P_{14})$$

therefore we have

$$P_{14} = 1/(u_1 T_{CPU})$$

(2)

It follows immediately that since there are N_{CPU} CPU service completions and N_{DIO} of them must have been due to an I/O

Analytical Modeling

request, then

$$P_{12} = N_{DIO}/N_{CPU} = N_{DIO} P_{14} \quad (3)$$

3.3.2 Program Swapping Behavior

The job swapping behavior is represented by the state-dependent routing probabilities P_{43} and P_{13} . Since there are several memory management schemes that can be implemented in an operating system, the expressions for P_{43} and P_{13} will vary from system to system. Chen derived expressions for these probabilities that are reasonable for the TOPS-10 Monitor on the DECsystem-10. Their derivations and the assumptions for their derivations are discussed below.

3.3.2.1 Derivation of P_{43}

The assumptions made for deriving P_{43} are as follows
(Ref 5:955)

1. Each job in the main memory is allocated the same amount of memory. Therefore, the maximum number of jobs that can be allocated in the main memory simultaneously is a constant and denoted by A.

$$A = \boxed{M/J}$$

Analytical Modeling

This assumption is justified since main memory is partitioned into a fixed number of parts and interactive jobs usually will fit in the same number of partitions.

2. Among all jobs in the main memory, the jobs in the "think" mode have the highest priority to be swapped out if memory space is needed for the jobs to be swapped in. This assumption is justified since many computer systems use this memory management strategy (Ref 5). Jobs in the "think" mode are very unlikely to need CPU service in the near future. This is true since the average think time is measured in seconds, while most other activities in the system occur in the millisecond or even microsecond range. In the DECsystem-10, jobs contained in the TIOWQ correspond to jobs in the think mode.

Let n_i denote the number of jobs at node i. If the total number of jobs in the system is less than the number of jobs that can fit in memory ($N < A$), then no swapping activity is necessary. In most cases, though, N is greater than A , and so $N - A$ jobs cannot be in memory. Since the n_4 jobs in the think mode have the highest priority to be swapped out (Assumption 2), these will be swapped out until $N - A$ jobs are swapped or until all jobs in the think mode are swapped out. Therefore, the probability that a job needs to be swapped in at the beginning of an interaction, P_{43} , is equal to the probability that a job in the think mode is not

Analytical Modeling

in main memory. Chen uses the approximation

$$P_{43} = \begin{cases} (N-A)/n_4 & n_4 > N - A \text{ and } N > A \\ 0 & N \leq A \\ 1 & n_4 \leq N - A \end{cases} \quad (4)$$

and therefore the probability that a job does not need to be swapped in at the beginning of the interaction is

$$P_{41} = 1 - P_{43} \quad (5)$$

3.3.2.2 Derivation of P_{13}

To derive P_{13} , two more assumptions must be made (Ref 5:955)

1. P_{13} is linearly related to the difference between number of active jobs ($n_1+n_2+n_3$) and the maximum number of jobs (A) that can fit in memory, if that difference is positive. The difference, $n_1+n_2+n_3-A$, is the number of jobs which need memory but can not have it. When this difference is positive, then some of these jobs as well, as the jobs in the think mode, must be swapped out. When this difference is less than or equal to zero, P_{13} is zero. This models the TOPS-10 Monitors swapping behavior of choosing jobs in long-term wait queues, e.g. interactive jobs waiting for terminal response or batch jobs waiting for a tape to be mounted, over jobs that are in the processing queues and more likely to run.

Analytical Modeling

2. In the worst case, the maximum number of times that a job will be swapped out before the end of the interaction is one. This assumption represents the ICPT's effect of locking the job in memory

Assumption 1 tells that P_{13} reaches a maximum when $n_1+n_2+n_3$ reaches its maximum value N. This maximum value can be determined using similar reasoning used in the derivation of Equation (3). By assumption 2 the maximum number of times a job can be swapped out is one. So the maximum probability of being swapped out is

$$P_{13} = N_{SWAP}/N_{CPU} = 1/(1/P_{14}) = P_{14}$$

Therefore, from assumption 1, P_{13} varies linearly from 0 to P_{14} as $n_1+n_2+n_3$ varies from A to N. Noting that P_{11} , P_{12} , P_{13} , and P_{14} must sum to 1, thus being bounded by $1-P_{12}-P_{14}$, and combining the above observations, we have

$$P_{13} = \begin{cases} \min[1-P_{12}-P_{14}, P_{14}((n_1+n_2+n_3-A)/(N-A))] & n_1+n_2+n_3 > A \text{ and } N > A \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

and therefore

$$P_{11} = 1-P_{13}-P_{12}-P_{14}$$

Analytical Modeling

to insure the probabilities sum to one.

3.3.2.3 Approximation Algorithm

The above formulation makes the probability transition matrix of Table 3-3 dependent on n_4 , i.e. the probability transition matrix is state-dependent. Unfortunately, this queueing network model has no exact solution known, so Chen proposes using the following approximation algorithm (Ref 5:956)

- Step 1 Assume an initial value for n_4 .
- Step 2 Use n_4 to calculate P_{43} and P_{13} by Equations (4) and (6).
- Step 3 Solve the model as a closed queueing network with fixed probabilities using Buzen's method.
- Step 4 If the new value of n_4 is very close to its previous value, the algorithm stops. Otherwise, Steps 2, 3, and 4 are repeated.

According to Chen, the algorithm will always converge since the sequence of computed values of n_4 is either monotonically increasing or decreasing and is bounded between 0 and N. Also, from Chen's computational experience, the algorithm converges to the same point, independent of its starting value.

Analytical Modeling

3.4 Chapter Summary

The following items are the important points brought out in this chapter

- Early models developed by Gordon and Newell, Jackson, and Buzen were restricted to single class networks with exponential servers. Baskett, et. al. consolidated later models which included multi-class networks and the method of stages to form one unified model.
- The Baskett, et. al. model provided a more accurate model of time-sharing systems, but the model still was not completely realistic; job swapping behavior is more complex and should depend on the main memory size, the number of jobs competing for memory, and the job sizes.
- Chen developed a model to make job swapping behavior depend on memory size, the number of jobs competing for memory, and the jobsizes, but failed to include the more general multi-class queueing networks. Also, his development only considered the single CPU and single disk system, and left the multiple CPU and multiple disk system as an extension.

In the next chapter we will look at the structure of the McKenzie FORTRAN program and how it was modified to include Chen's algorithm. Combining McKenzie's program with Chen's algorithm extended the Chen algorithm to multiple classes.

Chapter 4

Computer Implementation

The computational effort for the product form solution of a queueing network can be quite large for anything but very simple networks (see example in Chapter 3). Therefore, to compute the solutions to queueing networks containing many nodes and job classes, one must perform the computations for the product form solution on a computer.

The first section in this chapter describes the McKenzie computer program. This is followed by a section describing the modifications made to McKenzie's program to implement Chen's swapping model. After the Chen modifications are described, the next section discusses some of the pitfalls and problems encountered in this current effort to modify the program. The last section summarizes the chapter.

Computer Implementation

4.1 The McKenzie Program

In 1977 McKenzie developed a computer program to solve multi-class closed queueing networks with service centers having different service disciplines. This program provides a flexible tool to model interactive computer systems, but is limited to product form solutions of closed queueing networks. The program was modified to include Chen's model (see Chapter 3) to more accurately model job-swapping behavior.

4.1.1 Program Capabilities

The computer program is based on the model of Baskett, et. al. (Ref 2) and the computational algorithms of Wong (Ref 30). Wong's computational algorithms implement a subset of the entire Baskett et. al. theory. The limitations of the model are

- No outside arrivals can enter the network, i.e. the model is a closed queueing network.
- Jobs cannot transition from one job class to another at a service center.
- The third type of state-dependent service rate is not contained in the model. (see Chapter 3 for the types of state-dependent service rates)

Computer Implementation

4.1.2 Program Structure

McKenzie's computer program was written in FORTRAN IV. The program is modular in nature. It is divided into five subroutines called from the main program. The names of the subroutines and their functions are described below

INDATA	Reads the data necessary to run the program. The characteristics of the time-sharing computer system being modeled are input in this routine.
FUNCT	Calculates the values for the functions in the product form solution for each node in the network. These are used to calculate the joint and marginal probability density functions for each node.
NORMAL	Calculates the normalization constant for the model. The normalization constant assures that the system state probabilities sum to one.
MARGIN	Calculates the marginal probability density functions for each node in the network.
EXPECT	Calculates the probability that there are k users of class j at a node and the total probability over all classes of jobs that there are k users at a node. Using these probabilities, it then calculates node utilizations by job class, expected queue length probability distribution and their expected values by job class, mean time spent at each node by each job class, and system response times by job class.

In addition to these programs written by McKenzie, five other routines peculiar to the CDC computer at AFIT are used

WRITMS A random access routine to write records to mass storage.

Computer Implementation

READMS	A random access routine to read records from mass storage.
OPENMS	Initializes a file for random record reads and writes.
CLOSEMS	Updates the master index in the file after all records have been read or written.
LEQTLF	A subroutine that solves simultaneous linear equations.

The documentation on the random record routines is contained in reference 7. LEQTLF documentation is contained in reference 13.

The WRITMS is used at the end of INDATA, FUNCT, NORMAL, and MARGIN to save the computed values for each node on a random access file for further processing by the next subroutine. The READMS is used at the beginning of FUNCT, NORMAL, MARGIN, and EXPECT to retrieve the results of the calculations for each node from the preceding routine in the program flow (see Figure 4-1). OPENMS and CLOSEMS are used at the very beginning of the program and the very end of the program. The calculations are done node by node since the memory requirements necessary to store the entire network's state space would exceed the CDC computers memory resources for all except trivially small networks.

The subroutine LEQTLF is used in INDATA to compute the node departure rates using the probability transition

Computer Implementation

matrices and node service times for each class.

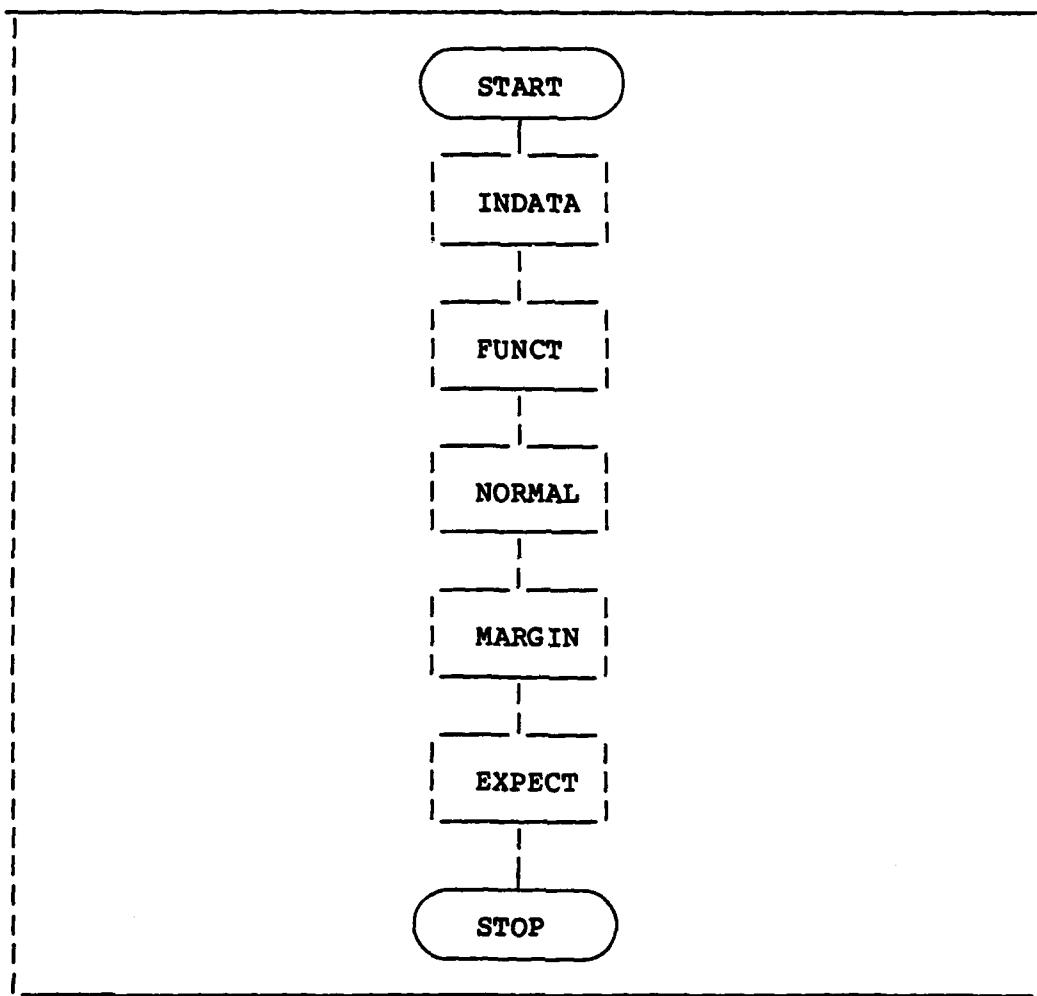


Figure 4-1 McKenzie Program Flow Chart

Computer Implementation

4.1.3 Program Inputs

The program inputs and formats are described in detail in Appendix C of McKenzie's thesis and will not be discussed here.

4.2 The Chen Modification

4.2.1 Program Structure

To implement Chen's algorithm, McKenzie's program had to be modified to

1. Read in the system configuration data necessary to use Chen's algorithm.
2. Compute the initial probability transition matrix for each class of job using the input data and Chen's formulae discussed in Chapter 3. The initial probability transition matrix is computed using Chen's formulae and then converted to the transition matrix for the classical swapping model.
3. Recompute the probability transition matrix for each class based on results from the previous iteration. Use this new probability transition matrix to compute the product form solution to the corresponding queueing model.
4. Stop iterating if the solution has converged, i.e. the absolute difference between the number of jobs in the think mode calculated in the previous iteration compared to the current iteration is within a specified tolerance, or if the maximum number of iterations allowed has been exceeded.

Computer Implementation

These modifications were localized to the main program and two subroutines in order to retain as much of McKenzie's structure as possible. INDATA was almost completely rewritten in order to read in the variables in Chen's model. One major benefit in this modification is that input to the model is now read in freefield format, i.e. data does not have to appear in certain columns in the data file. This will decrease the chance of human error when typing in the input to the model.

EXPECT was slightly modified in order to stop it from printing the results of every iteration. Now the results are printed for the first iteration, every fifth iteration, and then the final iteration.

The major changes to the McKenzie program to include Chen's algorithm were incorporated in the subroutines CHEN75 and COMPUTE. At the beginning of each iteration, CHEN75 is called, which in turn calls COMPUTE to recompute the probability transition matrix for each class of job in the network using the formulae from Chen's model.

Chen's model was also extended to include multiple disk drives. Given that a job from a particular class needs a disk I/O, one must define the disk access probabilities for

Computer Implementation

each disk in the system. Defining N_{DISKS} to be the number of disks in the system, and $P_{DIO}(i)$ the probability of performing the disk I/O on the i^{th} disk, then the probability transition matrix is modified as shown below

P_{11}	$P_{12} * P_{DIO}(1) \dots P_{1(N_{DISKS}) + 1} * P_{DIO}(N_{DISKS})$	P_{13}	P_{14}
1	0	0	0
.
.	(N_{DISKS} times)	.	.
.	.	.	.
1	0	0	0
P_{41}	0	P_{43}	0

To conform to normal probability transition matrix notation, the subscripts are renumbered. For example, in the case of a system with six disks, the subscripts are changed so that P_{13} becomes P_{18} , P_{14} becomes P_{19} , P_{41} becomes P_{91} , etc.

The changes described above are illustrated in the new flow chart for the program shown in Figure 4-2.

4.2.2 Program Inputs

The input to the new program is simpler now, since all input data values are read in freefield, instead of requiring the values to be in particular columns. All that

Computer Implementation

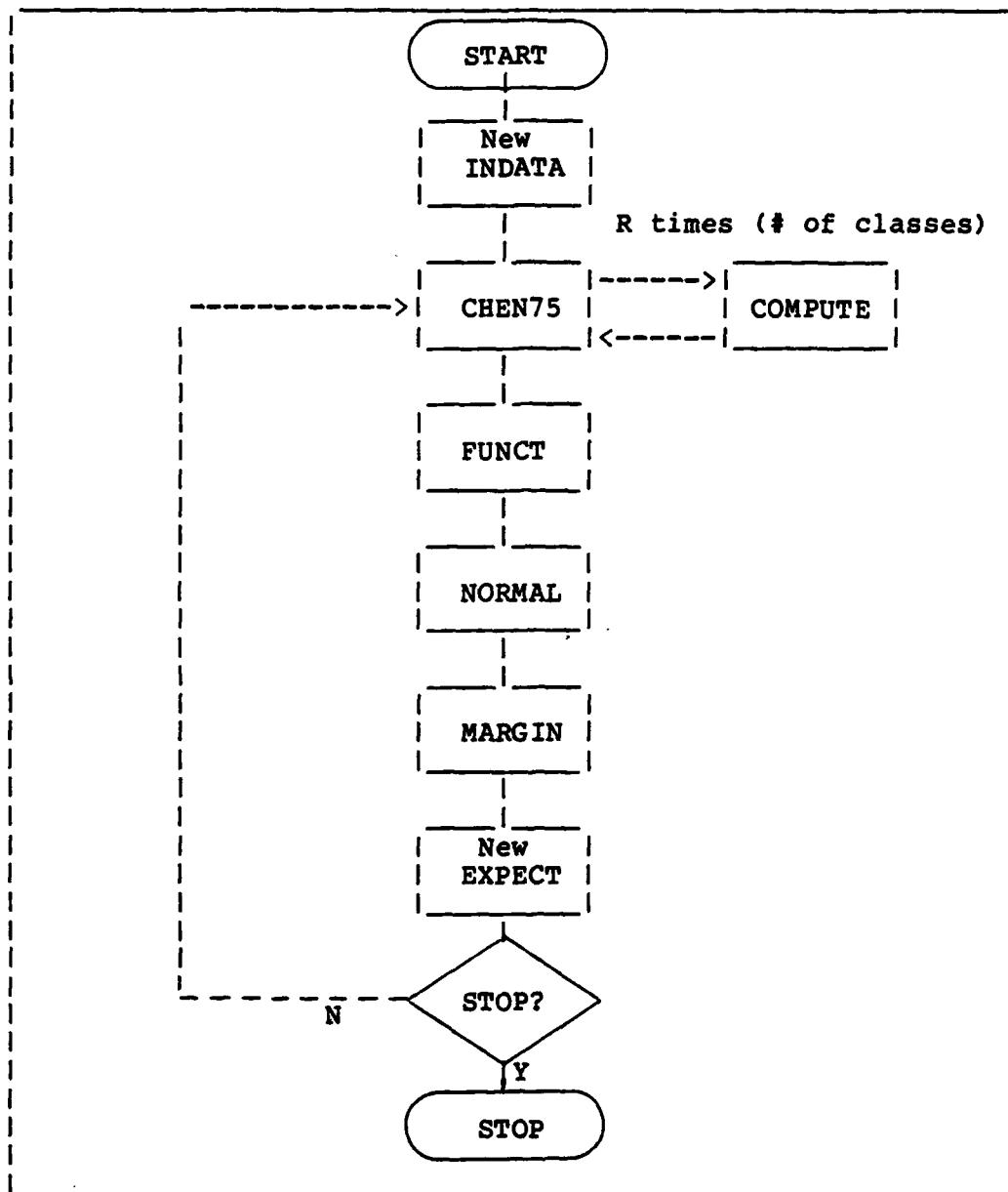


Figure 4-2 New Program Flow Chart

Computer Implementation

is necessary is that the input data values be separated by at least one space. The data could be entered on one line until that line is filled up and continue in that fashion for subsequent lines. Instead, it is recommended that the data be entered in the following manner (the names in capital letters are the FORTRAN variable names)

- **Network description.** Enter the number of nodes and number of job classes in the network.

NODES, ITYPES

- **Computer system configuration.** Enter the number of disk drives in the I/O subsystem and the size of main memory available for user jobs.

NDISKS, SIZEMEM

- **Workload configuration.** If possible, enter the following workload configuration parameters on separate lines

1. Disk access probabilities by class for each disk. If possible, put the probabilities for each disk on separate lines. There would then be NDISKS lines.

DISKPRB(I,J) (probability that a member
of the j^{th} class
accesses the i^{th} disk)

2. Job class characteristics. Put the characteristics for each class on a separate line. This includes the number of jobs in the class, the CPU time per interaction (T_{CPU}), the number of disk I/O's per interaction (N_{DIO}), and average job size of the class. There should be ITYPES lines of job class characteristics.

Computer Implementation

(NUSERS(J), TCPU(J), DIO(J), SIZEJOB(J))

- **Node characteristics.** The node characteristics include the following

1. Node type (FCFS (NODETYP(I)=1), PS (NODETYP(I)=2), IS (NODETYP(I)=3), and LCFS (NODETYP(I)=4)).
2. Types of state-dependent service rates can be None (IDEP(I)=1), number of total jobs at node (IDEP(I)=2), number of jobs in a particular class at the node (IDEP(I)=3).
3. Service rate for job class J and node I (SERVICE(I,J)).
4. State-dependent service rates. If IDEP(I) was not 1, then the state-dependent service rate data must be entered. The entered values represent the ratio of the service time when there is a total of one job at the node (IDEP(I)=2) or when there is one job of a particular class (IDEP(I)=3). For example, to model two identical servers, one would set IDEP(I)=2 and enter

1.0 2.0 2.0 ... 2.0

into array (DEP(I,1,K), K=1, MAXUSER) The number of entries is equal to the maximum number of users of all types in the network (MAXUSER).

4.3 Programming Notes for Future Modification

If future thesis students decide to work with the program used in this thesis, the student should be aware of some possible pitfalls. The problems encountered when

Computer Implementation

making changes to the program are described in the following paragraphs.

The original McKenzie program depended on the operating system to initialize the variables in the model to zero. Unfortunately, the CDC NOS/BE operating system no longer zeros out the memory locations before loading a program, but instead sets them to a special value signifying an indefinite value. In the process of modifying the program, all uninitialized variables were initialized to zero for each iteration. Some variables may have slipped by, though, so if any "Error Mode 4-- Indefinite Value" messages appear after modifying the program, check for uninitialized variables.

In order to have a variably dimensioned array passed to a FORTRAN IV subroutine, one must not only pass the variable dimension value, but also the memory space (in the form of an array from the main program). That is the reason the subroutine parameter lists are so large. Unfortunately, if one passes incorrect dimensions of a passed array to the subroutine, no syntax error is generated. Instead, when the program is run, it will produce one of possibly several mode errors. Therefore, when making any changes to the dimensions of the arrays in the main program, be sure to

AD-A115 565 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL--ETC F/G 9/2
MULTI-CLASS ANALYTICAL MODELS OF THE DECSYSTEM-10 JOB-SWAPPING --ETC(II)
DEC 81 M H COX
UNCLASSIFIED AFIT/GOR/MA/81D-4 NL

202
AFIT-GOR/MA/81D-4

END
DATE
PRINTED
7 82
DTIG

Computer Implementation

change the appropriate variables (MAXNODE, MAXTYPE, MAXUSER, MAXSTAT, MAXUSE1, and ISIZE3) to match the true dimensions.

Finally, one must be careful that the inputs for the model are consistent. Specifically, a common and somewhat obscure error is to define a probability transition matrix that unintentionally isolates a node from the network. This causes the departure rate from that node to be zero. When the subroutine FUNCT is called to compute the functions used in the product form solution for that transition matrix, that zero value is used as a base in an exponential expression (line 34) which can have a zero exponent. Since zero to the zero power is undefined, another mode error is produced; this time "Error Mode 2--infinite operand". Therefore, always make sure the probability transition matrix used in the computations is well-defined.

4.4 Chapter Summary

In this chapter, we have reviewed the basic structure of the McKenzie program and the functions of its five subroutines. The modifications made to this program were identified, and the new subroutines CHEN75 and COMPUTE discussed. The modifications were made with the objective

Computer Implementation

of retaining as much of the original McKenzie model as possible. All of the changes to the original program were in the main program, INDATA, and EXPECT. The addition of Chen's algorithm was accomplished by including CHEN75 and COMPUTE. Finally, some programming notes on this author's experience with modifying the program were presented. Briefly, these included

1. If any "Error Mode 4-- Indefinite Value" messages appear after modifying the program, check for uninitialized variables.
2. When making any changes to the dimensions of the arrays in the main program, be sure to change the appropriate variables (MAXNODE, MAXTYPE, MAXUSER, MAXSTAT, MAXUSE1, and ISIZE3) to match the true dimensions.
3. Always make sure that the probability transition matrix used in the computations is well-defined.

With this modification to McKenzie's original program, we now have a new tool to model interactive computer systems which considers multi-class job-swapping behavior as a function of main memory size, the number of jobs competing for memory, and the job size. Chapter 5 presents the results of using the multi-class Chen swapping model and the classical swapping model on three hypothetical workload models.

Chapter 5

Analytical Modeling Results

This chapter contains the results of modeling the DECsystem-10 using the single-class and multi-class Chen swapping model on a realistic, though hypothetical interactive/batch workload configuration. The workload was modeled using three alternate approaches.

The first section discusses the three alternate approaches used to model the workload configuration. The next section describes the system configuration used to model the DECsystem-10. The third section presents the results of two model comparisons: Chen's swapping model versus the classical model and the multi-class versus the single-class Chen swapping model.

5.1 Modeling Interactive/Batch Workloads

When a CPE analyst is faced with the task of analyzing the performance of an interactive computer system containing

Analytical Modeling Results

batch jobs with an analytical model, he has the following alternatives

1. Characterize the interactive and batch job classes separately and use a multi-class queueing model.
2. Use a single class model and make one of following commonly used approximations
 - Create a single, average job class by aggregating the characteristics of the individual classes.
 - Assume the impact of batch jobs is minimal, and use a single class with the characteristics of the interactive jobs. This approach was used by Chen in his original work.

5.2 DECsystem-10 System Configuration Parameters

Because of the large number of system configuration parameters which can be varied within the models, an infinite variety of system configurations could be analyzed. A subset of them were arbitrarily fixed in order to compare modeling approaches. These configuration parameters can be divided into two types: hardware and workload.

5.2.1 Hardware Parameters

The hardware configuration parameter values were chosen to model as closely as possible the Avionics lab

Analytical Modeling Results

DECsystem-10 hardware configuration. These hardware parameters are given in Table 5-1. The hardware configuration includes two CPU's, six disk drives within the I/O subsystem and one swapping device.

Table 5-1 System Configuration Parameters

<u>Hardware Parameters</u>		
Number of Disks = 6		
Number of CPU's = 2		
Number of Swapping Devices = 1		
<u>Workload Parameters</u>		
<u>Interactive</u>		<u>Batch</u>
Number of Jobs	30	10
Job Size (K)	5	10
CPU quantum (sec)	.0125	.015
Disk Access Probabilities	Equal	Equal
Disk Service Times (sec)	.075	.075
Long-term wait Service (sec)	8.0	1.0
Job Swap Time (sec)	.1	.2
T _{CPU} (sec)	.1	.5
N _{DIO}	4.0	5.0

Analytical Modeling Results

5.2.2 Workload Parameters

The workload configuration parameter values shown in Table 5-1 were selected to model typical interactive and batch jobs. These values are within the range of values used by Chen to assure their reasonableness.

Additional knowledge of the DECsystem-10 and of interactive and batch jobs in general were used to set the relationships between the magnitudes of the workload parameters. These are described below for each workload parameter.

Job Size The above job sizes are within the range of job sizes used by Chen. The interactive job size was chosen smaller than the batch jobs since this is true on the average. Interactive jobs include many editing, compiling, and other utility programs which have small memory requirements. Batch jobs, on the other hand, include the running of large application packages such as SPSS, SLAM, etc.

CPU Quantum This value corresponds to the jiffy described in Chapter 2. The maximum CPU quantum is one jiffy (.01667 seconds), disregarding overhead of the operating system. When overhead rates of 10-25% are included, the CPU quantums have the values of those above. Interactive jobs usually have more overhead, since they deal more with system utilities. Batch jobs usually are large, number-crunching applications, and utilize little overhead.

Disk Access Probabilities

Since no other information was available these probabilities were arbitrarily chosen to be equal for all disks and for all job

Analytical Modeling Results

classes to simplify the analysis.

Disk Service Times

These times were again arbitrarily chosen to be equal for all disks and for all job classes. The value of the service time is consistent with the times used by Chen.

Long-term Wait Service Time

The value chosen for the interactive job class represents the "think" time for the user (the time to make a response to the computer) as well as the other long-term wait service times common to both interactive and batch jobs, e.g. command wait or DAEMON wait described in Chapter 2. The value selected is consistent with the values used by Chen.

Since jobs in the batch job class do not have a "think" time, but do enter other long-term wait states common to both interactive and batch jobs, the batch job class was assigned a long-term wait service. No information was available to determine typical values for long-term wait service times for batch jobs. The batch long-term wait service time was chosen to be significantly less than the interactive long-term wait service time since the long-term wait states possible for batch jobs require considerably less time to satisfy, and are therefore, much faster than the "think" time of an interactive user.

Job Swap Time

The values selected are consistent with the values used by Chen. The swap time for the batch job class should be larger since there is more batch job memory to swap. The batch job swapping time was arbitrarily doubled to take this into account.

T_{CPU}

The values selected for the CPU time per interaction are consistent with the values used by Chen. The CPU time per interaction for interactive jobs is lower than the value for batch jobs because interactive jobs are generally less CPU-intensive than batch jobs.

N_{DIO}

The values selected for the number of disk

Analytical Modeling Results

I/O's per interaction are reasonable considering the values used by Chen. Since the number of disk I/O's per interaction for interactive jobs is lower than the number of disk I/O's per interaction for batch jobs, the natural conclusion to draw is that the batch jobs are more I/O-intensive than the interactive jobs (usually the opposite is true). A more insightful approach shows that the interactive jobs are the more I/O-intensive. Equation 3-1 gives the formula for the number of CPU quantums received per interaction, N_{CPU} . This formula and Equation 3-3 yield a value of 8 and 33 1/3 for interactive and batch jobs, respectively. Now the values of N_{DIO} are more meaningful. In the case of the interactive jobs, half of the CPU quantums were terminated due to disk I/O requests, while in the case of the batch jobs, only 15 percent of the CPU quantums were terminated due to disk I/O requests. Therefore in this workload configuration, the interactive jobs are the more I/O-intensive.

5.2.3 DECsystem-10 Swapping Model

The multi-class Chen swapping model derived from the above system configuration is shown in Figure 5-1. Note that the probabilities and service times for the network are class dependent (r). Node 1 corresponds to the two CPU's; nodes 2 through 7 represent the I/O subsystem containing the 6 disk drives; node 8 models the swapping device; and node 9 represents the long-term wait service node.

The following are definitions of the important probabilities in the network

P_{91r} P_{91r} is the probability that the job in class

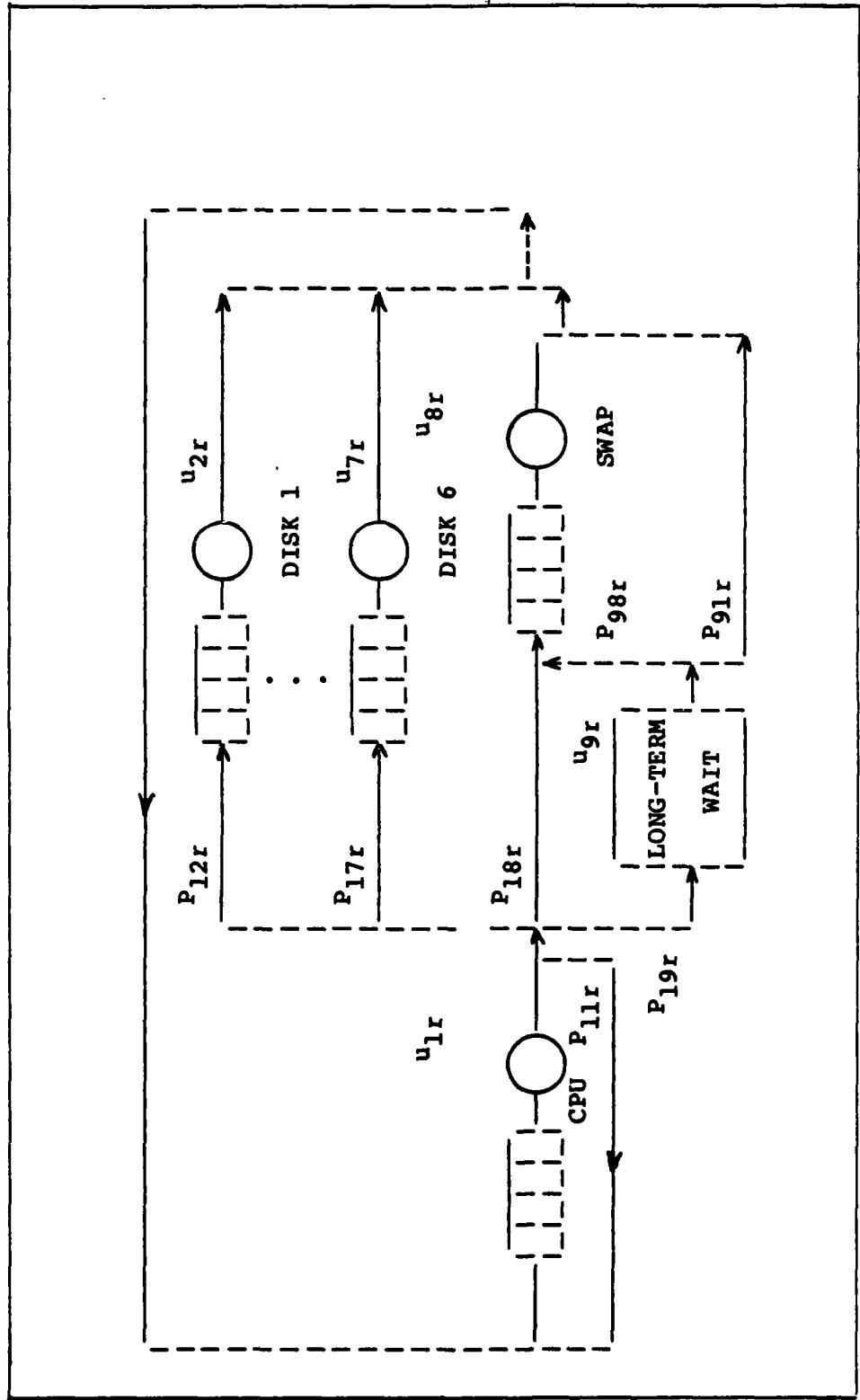


Figure 5-1 DECsystem-10 Swapping Model (For Class r)

Analytical Modeling Results

P_{98r} leaving a long-term wait state (the "think" mode for interactive jobs) is already in memory and can proceed directly back to the CPU (analogous to P_{41} in Chen's original model).

P_{98r} is the probability that a job in class r leaving a long-term wait state (the "think" mode for interactive jobs) must be swapped into memory (analogous to P_{43} in Chen's model).

P_{18r} is the probability that a job in class r finishing its CPU quantum, must be swapped out to make room for another job which is ready to execute, but has been waiting for memory (analogous to P_{13} in Chen's model).

P_{1jr} , for $j = 2, 3 \dots 7$, are the probabilities of a job in class r going to the various disks in the I/O subsystem from the CPU (analogous to P_{12} in Chen's model).

5.3 Model Comparisons

5.3.1 Definition of Performance Measures

The following are definitions of the performance measures considered in these comparisons (Ref 29:10-12)

CPU utilization

The percent of time the CPU is processing a job.

Swapper utilization

The percent of time the swapping device is swapping in or out a job.

Analytical Modeling Results

Response Time Elapsed time between entering the last character of a request at a terminal and receiving the first character of the response.

5.3.2 Tabular Results

The results of the models are shown in Tables 5-2 through 5-4. Table 5-2 contains the performance measure results for the classical swapping model with the three workload cases. Table 5-3 contains the results for the Chen swapping model with the three workload cases. Table 5-4 shows the behavior of the probability transition matrix for different main memory

Table 5-2 Classical Swapping Model Results

<u>Performance Measures</u>			
Workload Cases	CPU Utilization (%)	Swapper Utilization (%)	Response Time (sec)
Two Class	100.00	33.46	6.750
Average	99.98	33.23	8.756
No Batch	81.05	26.97	3.122

<u>Important Probability Matrix Values</u>			
	P_{11r}	P_{18r}	P_{91r}
Two Class	.466/.933	0	0
Average	.699	0	0
No Batch	.466	0	0

Analytical Modeling Results

Table 5-3 Chen Swapping Model Performance Results

Memory (K)	Workload	CPU Utilization (%)	Swapper Utilization (%)	Response Time (sec)
240	Two Class	98.78	6.44	.828
	Average	70.95	4.00	1.083
	No Batch	-	-	-
200	Two Class	98.58	25.72	.847
	Average	70.80	16.00	1.107
	No Batch	-	-	-
150	Two Class	97.94	51.14	.895
	Average	70.51	32.00	1.154
	No Batch	-	-	-
100	Two Class	95.84	75.24	1.022
	Average	70.05	48.00	1.227
	No Batch	30.11	12.50	.504
50	Two Class	89.15	93.06	1.400
	Average	69.24	64.01	1.350
	No Batch	29.95	25.00	.557
40	Two Class	87.07	95.10	1.521
	Average	68.95	67.89	1.407
	No Batch	29.91	27.50	.570
30	Two Class	85.12	96.48	1.640
	Average	68.65	71.28	1.458
	No Batch	29.87	30.00	.583
20	Two Class	84.21	96.99	1.697
	Average	68.49	72.81	1.484
	No Batch	29.83	32.50	.598
10	Two Class	82.52	97.76	1.804
	Average	68.18	75.54	1.536
	No Batch	29.78	34.99	.613

Analytical Modeling Results

Table 5-4 Important Probability Matrix Values

Memory (K)	Workload	P_{11r}	P_{18r}	P_{91r}
240	Two Class	.374/.820	0	.934
	Average	.521	0	.941
	No Batch	-	-	-
200	Two Class	.374/.820	0	.735
	Average	.521	0	.764
	No Batch	-	-	-
150	Two Class	.374/.820	0	.468
	Average	.521	0	.526
	No Batch	-	-	-
100	Two Class	.374/.820	0	.188
	Average	.521	0	.282
	No Batch	.374	0	.641
50	Two Class	.359/.816	.015/.003	0
	Average	.521	0	.025
	No Batch	.374	0	.286
40	Two Class	.351/.814	.023/.005	0
	Average	.519	.002	0
	No Batch	.374	0	.214
30	Two Class	.344/.812	.030/.007	0
	Average	.515	.006	0
	No Batch	.374	0	.141
20	Two Class	.341/.811	.033/.008	0
	Average	.513	.008	0
	No Batch	.374	0	.068
10	Two Class	.335/.810	.039/.009	0
	Average	.510	.011	0
	No Batch	.374	.000	0

Analytical Modeling Results

sizes and workload cases. The Chen versus classical swapping model is discussed first to verify the Chen swapping algorithm results. Then a comparison between the two class model versus the single-class models is presented.

5.3.3 Chen's Versus the Classical Swapping Model

The following paragraphs consider the impact of using Chen's swapping model versus using the classical swapping model. The performance measures will be utilization of the CPU and the swapping device, and the response time received by the interactive users. The results shown are only for the two class model. Similar results were achieved with the average class model and the no batch model.

5.3.3.1 Probability Structure

The primary difference between the classical swapping model and Chen's swapping model is that Chen's probability transition matrix is dynamic, i.e. the probabilities in the matrix are dependent on the main memory size, the number of jobs competing for memory, and the job sizes.

The Figure 5-2 is a plot of the data in Table 5-4 for the two class workload model. It illustrates the dynamic

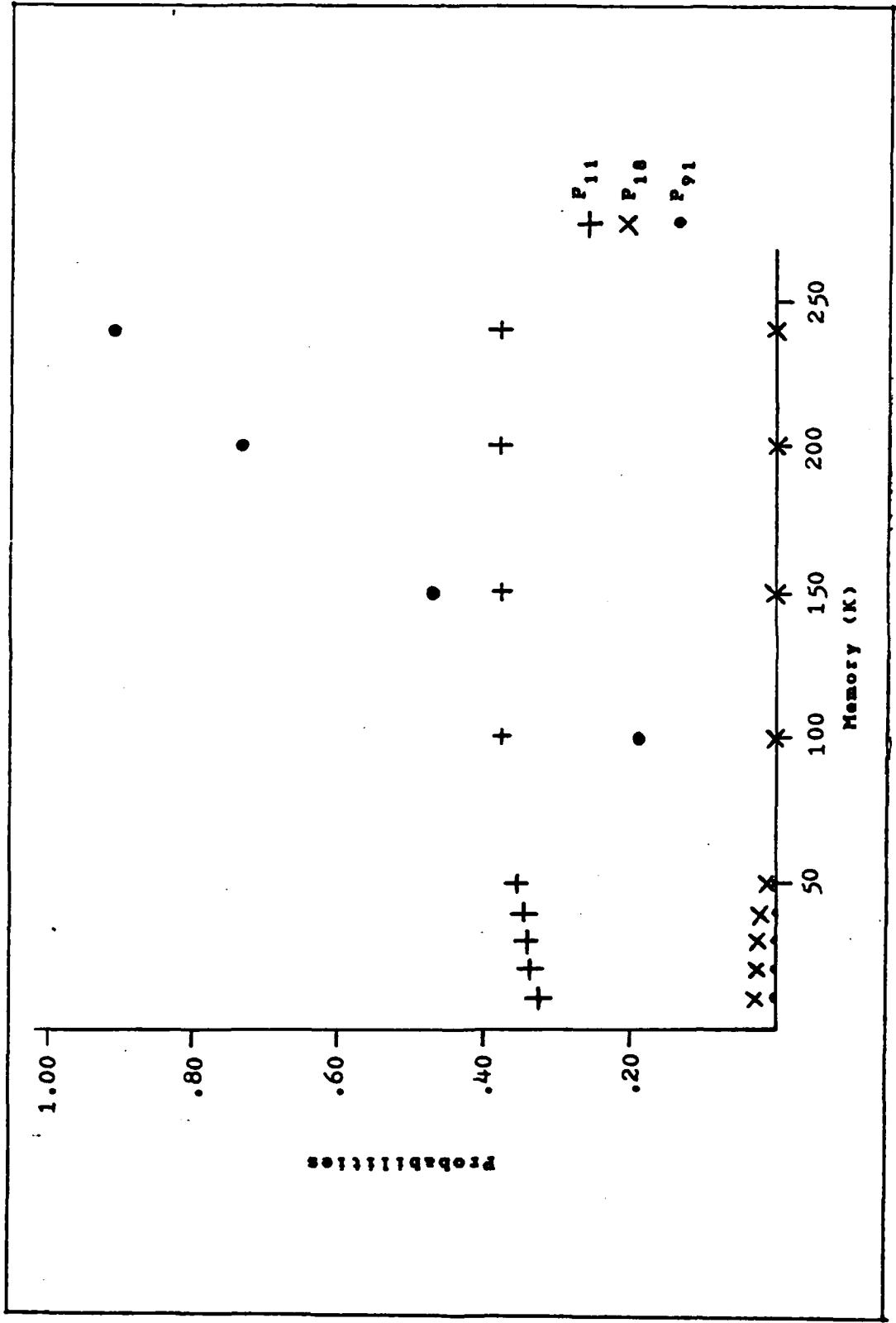


Figure 5-2 Probability Structure (Two Classes)

Analytical Modeling Results

relationship between the state-dependent probabilities and main memory size produced by the two class workload model. In the classical model P_{18r} and P_{91r} would be zero and P_{11r} would be constant. With Chen's model, the probabilities have some interesting properties

- P_{91r} behaves "classically" ($P_{91r}=0$) from zero to approximately 70K of main memory. At that point the Chen approximation takes effect and P_{91r} steadily increases to a probability of one at 250K of main memory.
- P_{11r} and P_{18r} behavior is reversed. For ranges of main memory between 70K and 250K, the probabilities behave "classically" ($P_{11r}=\text{constant}$, $P_{18r}=0$). Between 10K and approximately 70K of main memory, the Chen approximation takes effect. In this range, P_{11r} is inversely proportional to P_{18r} , i.e. P_{11r} 's gain is P_{18r} 's loss.

Similar phenomena occur in the average class model and the no batch model.

The rationale for the behavior of P_{91r} is as follows. For small amounts of main memory, all jobs in long-term wait will be swapped out to make room for other jobs waiting for memory. Hence, whenever a job gets blocked at the CPU and goes to a long-term wait state, it will be swapped out and must therefore always be swapped back in when it finishes its long-term wait service. For large amounts of main memory, there is enough memory to satisfy all memory demands as well as keep all jobs in a long-term wait state in

Analytical Modeling Results

memory. As main memory size approaches 250K, an increasing number of jobs in a long-term wait state may stay in memory. Therefore, this increases the probability that a job leaving a long-term wait state will already be in main memory and need not be swapped in.

The rationale for the behavior of P_{11r} and P_{18r} is as follows. For large amounts of main memory, there is enough for all executable jobs to remain in memory. As main memory size gets smaller, a larger number of jobs in long-term wait are swapped out. This continues until all jobs in the long-term wait states swapped out. After all the inactive jobs have been swapped out, the Job Swapper must start swapping out jobs that are executable in order to make room for other jobs in the system. Therefore, jobs expiring their CPU quantum now have higher probabilities of being swapped out, instead of returning to the end of the CPU queue for more service (P_{18r} becomes larger).

5.3.3.2 Performance Predictions

The figures 5-3 a) through 5-3 c) are plots of the data contained in tables 5-2 and 5-3 for the two class model. The horizontal lines on each plot represent the values predicted by the classical model, while the data points

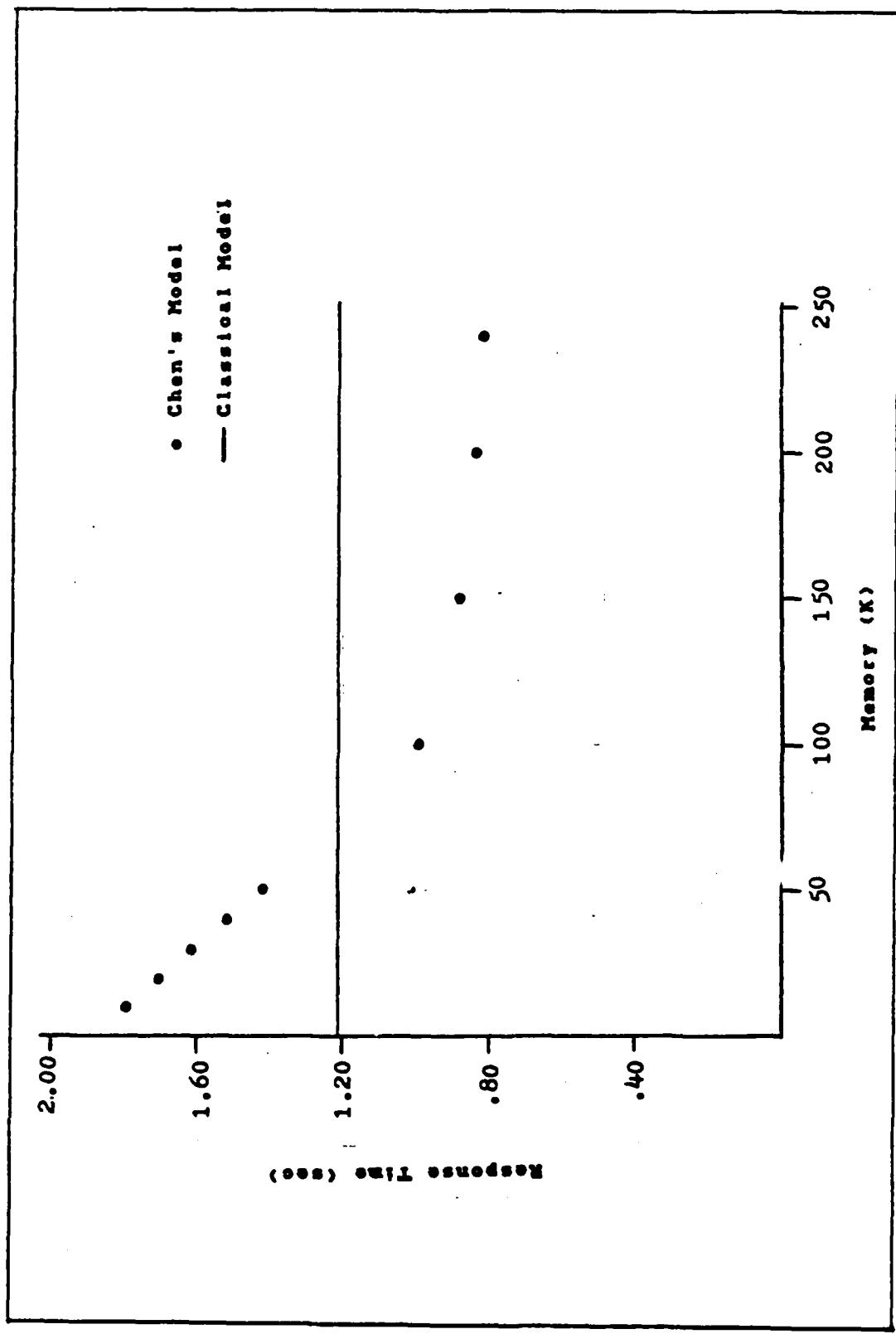


Figure 5-3 a) Two Classes

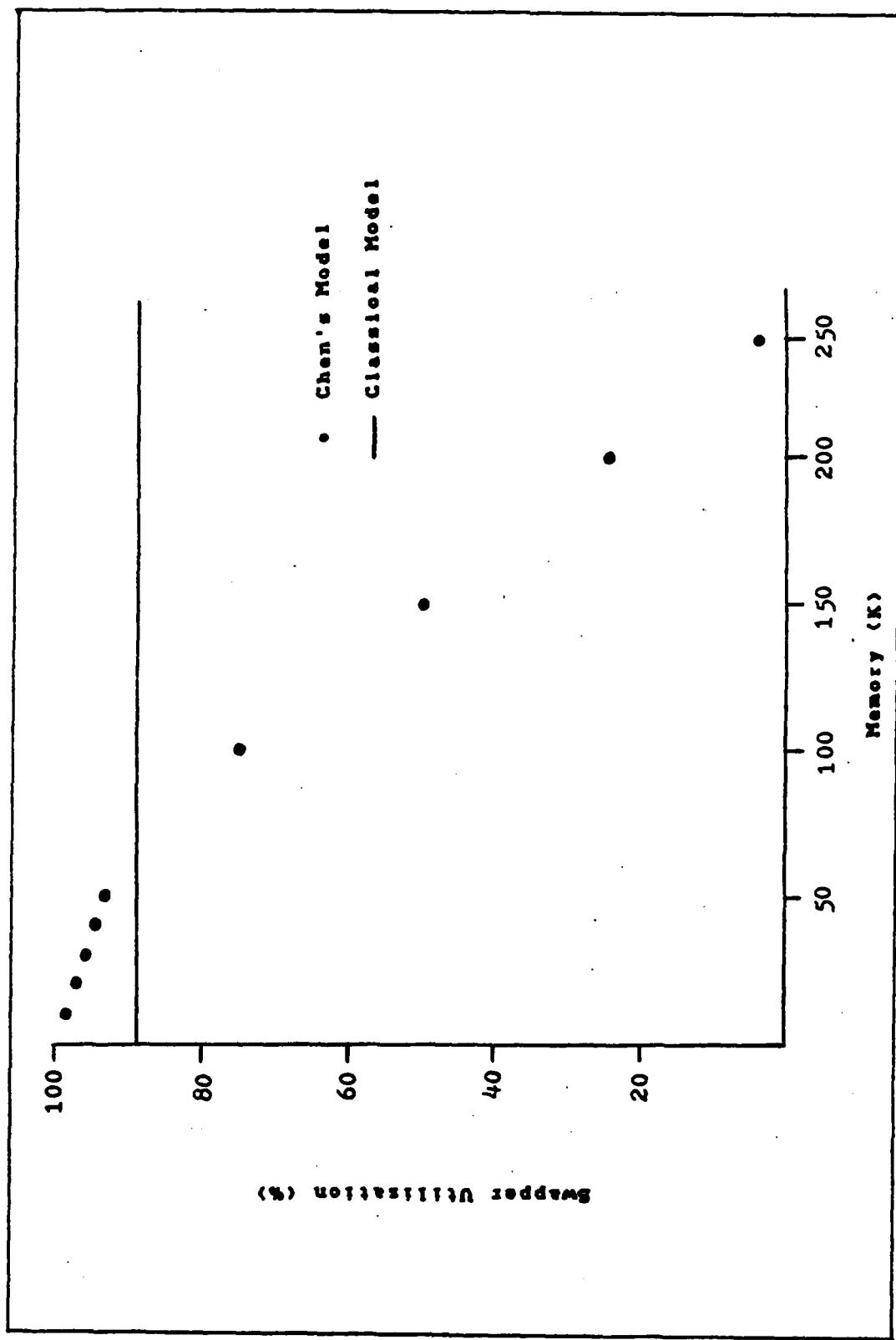


Figure 5-3 b) Two Classes

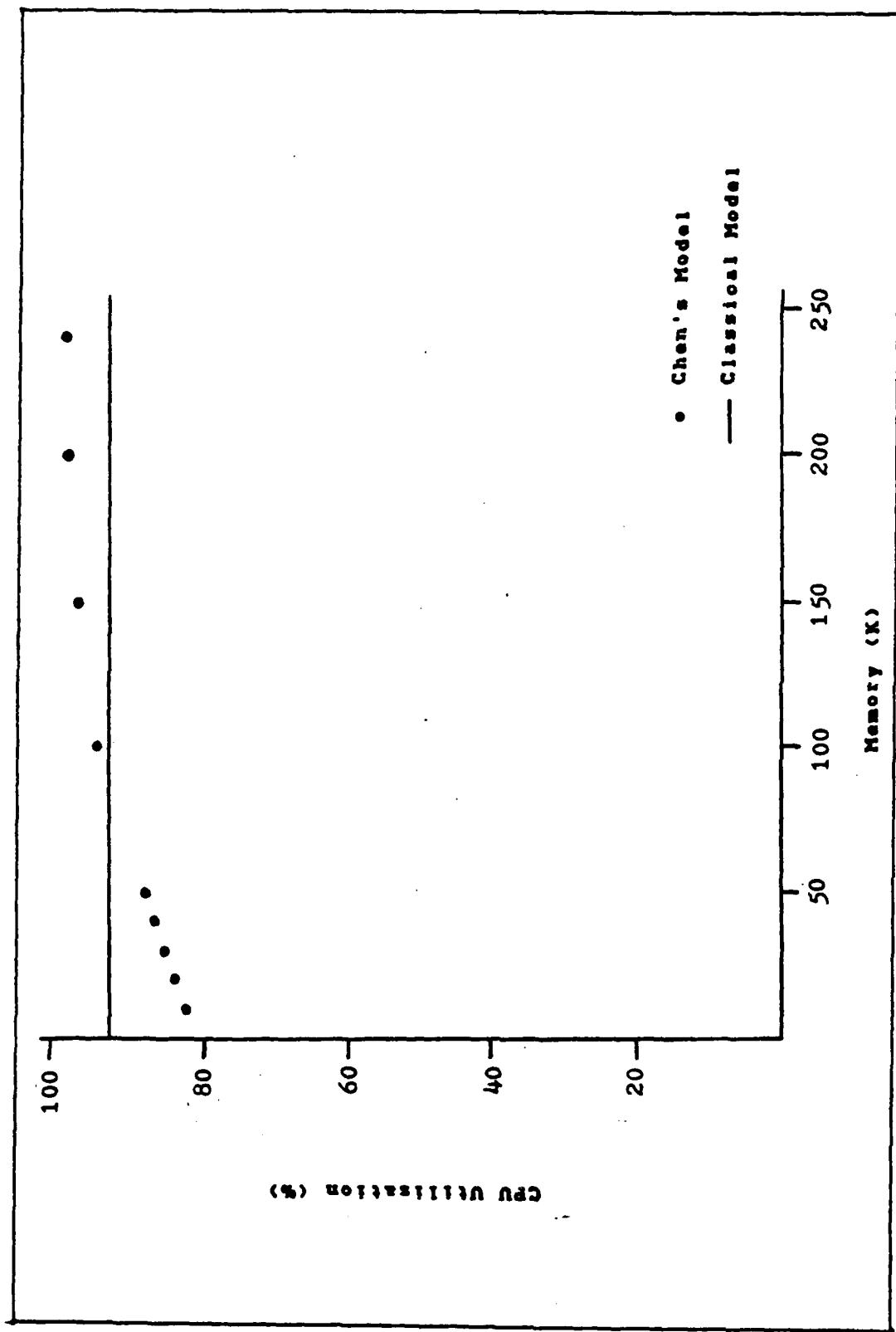


Figure 5-3 e) Two Classes

Analytical Modeling Results

represent the values predicted by the Chen model at each memory size. Note the sensitivity of the performance predictions due to the effects of main memory size in Chen's swapping model. Specifically, the following effects are modeled in the Chen swapping model

- In the range from zero main memory to the main memory size at the intersection of the Chen swapping model curve and the classical swapping model curve (approximately 70K for the two class model), the Chen model includes the effect of swapping out executable jobs on the performance measures (P_{19}). Compared to the classical swapping model, the Chen swapping model shows
 - * Increased response time and swapping utilization caused by increased swapping activity.
 - * Decreased CPU utilization caused by less jobs in memory competing for service from the CPU's.
- For the range of memory values between the point of intersection and enough memory to contain all jobs (250K for the two class model), the Chen model includes the effect of gradually having to swap in and out fewer jobs in long-term wait as opposed to the classical assumption that jobs in long-term wait are always swapped in. Compared to the classical swapping model, the Chen swapping model shows
 - * Decreased response time and swapping utilization caused by decreased swapping of the jobs in a long-term wait state.
 - * Increased CPU utilization caused by more jobs in memory competing for service from the CPU's.

5.3.4 Chen's Swapping Model: Multi-class Versus Single-class

The following paragraphs compare the results of the two class model to the average class and no batch class workload

Analytical Modeling Results

models when using Chen's swapping model. The performance measures considered will be the same as those used in comparing Chen's swapping model versus the classical swapping model: utilization of the CPU and the swapping device, and the response time received by the interactive users.

The figures 5-4 a) through 5-4 c) are plots of the data contained in Table 5-3 for the two class, average class, and no batch workload models. These plots reveal the wide disparity between the results obtained from a single class approximation when actually two classes of jobs exist. In particular, one can note the following results

- The no batch workload model seriously underestimates the response time of interactive jobs. For the average workload model, the response time is overestimated for main memory sizes which allow all active jobs to reside in memory. For main memory sizes below this value, response times are underestimated.
- The single-class workload models underestimate the CPU and swapper utilizations. This is especially true for the no batch workload model. As main memory sizes increases, the average workload model approaches the two class model.
- For all performance measures considered, the average workload model was closer to the two class workload model than the no batch workload model.

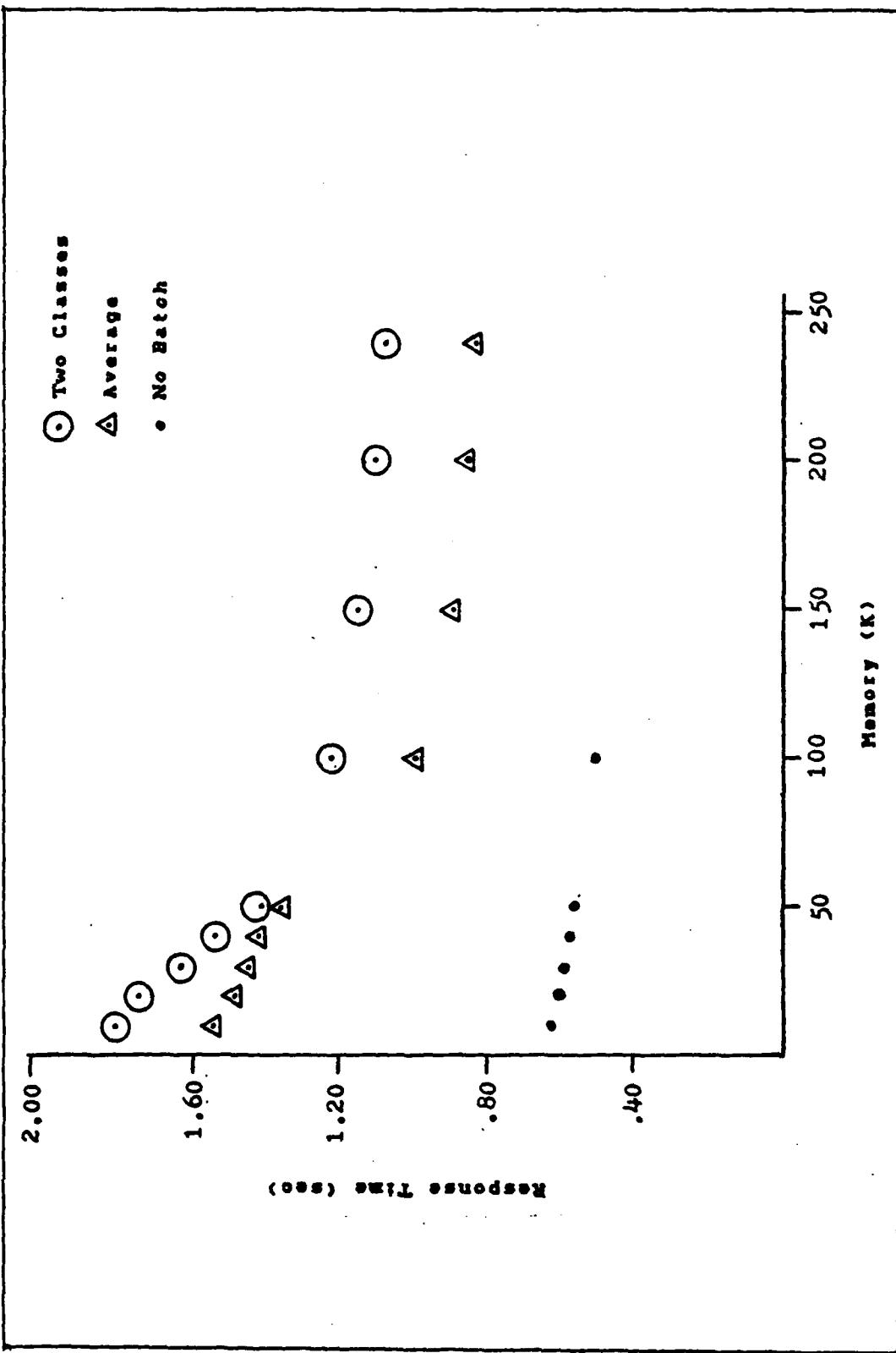


Figure 5-4 a) Chan's Model

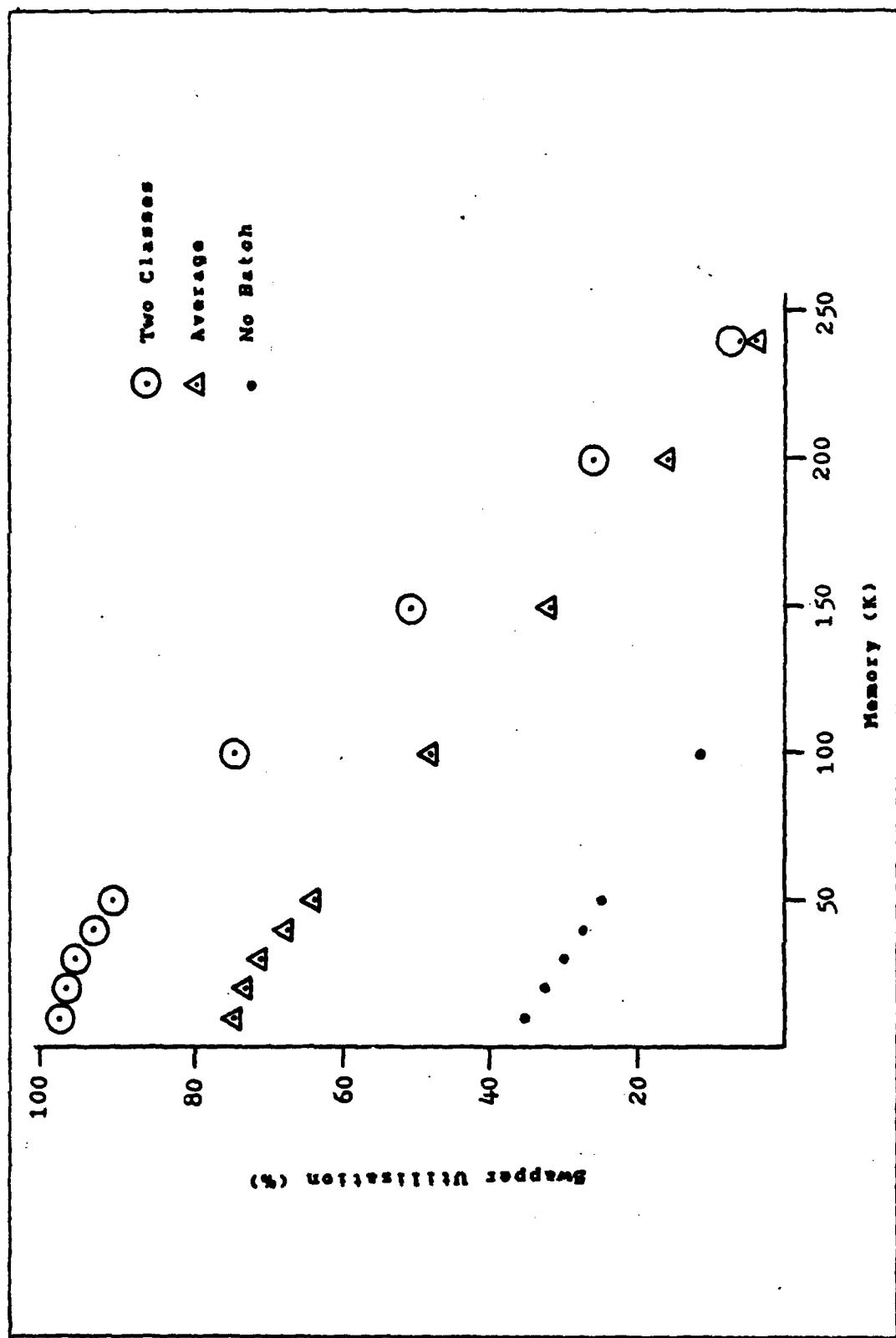


Figure 5-4 b) Chen's Model

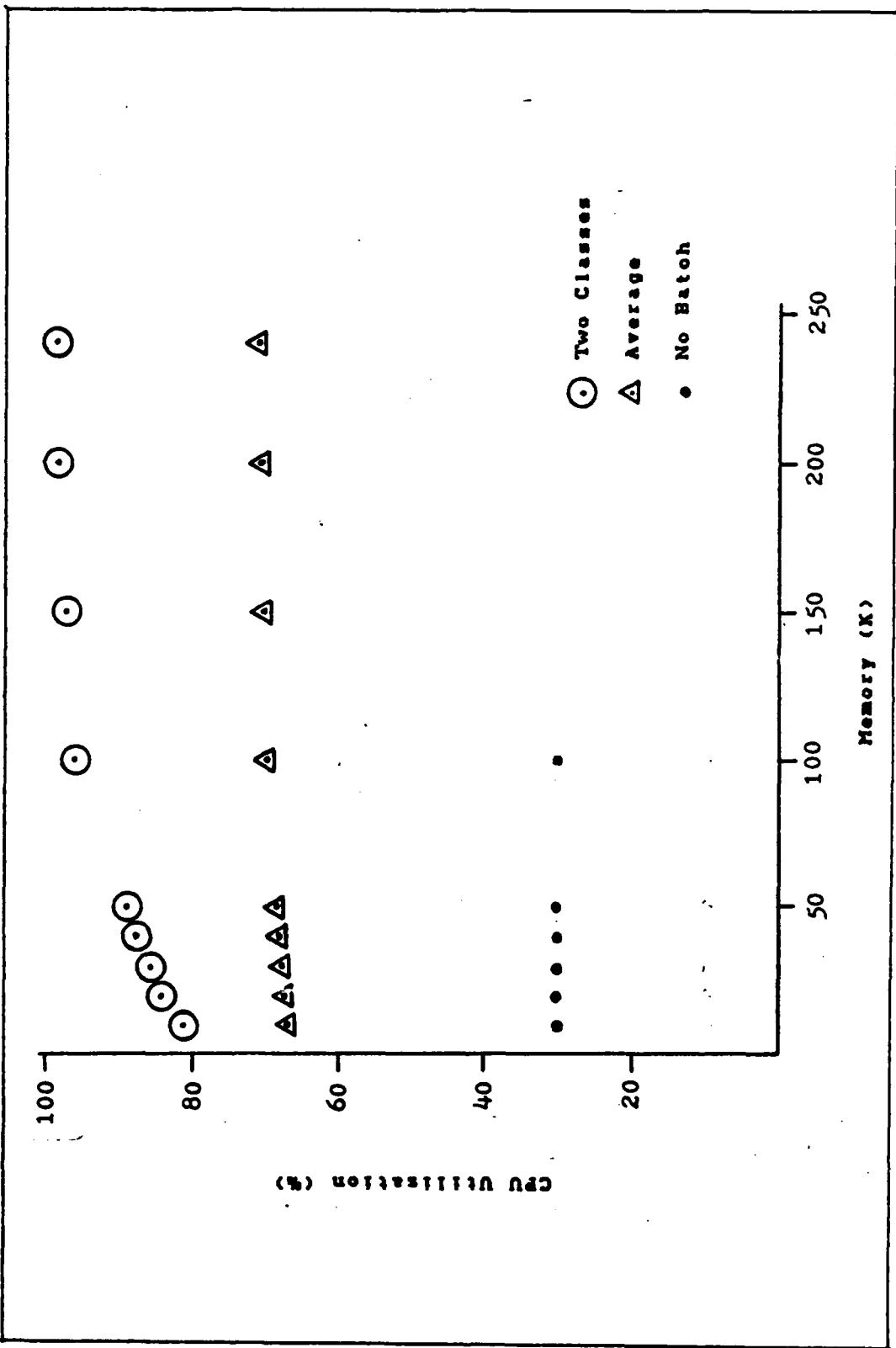


Figure 5-4 o) Chen's Model

Analytical Modeling Results

5.4 Chapter Summary

This chapter presented the results of modeling a computer system using the Chen model extended to multi-class models for three different workload cases: two class, average class, and no batch. The hardware and workload configuration parameters were chosen to as closely model the Avionics Lab system configuration as possible and to remain consistent with the workloads considered by Chen.

The results of the comparison between the Chen swapping model and the classical swapping model are summarized below

- For small amounts of main memory
 - * Longer response time and higher swapping utilization caused by increased swapping activity.
 - * Lower CPU utilization caused by more jobs waiting for swapping service and less jobs in memory competing for CPU service.
- For larger amounts of main memory
 - * Shorter response time and lower swapping utilization caused by decreased swapping of the jobs in a long-term wait state.
 - * Higher CPU utilization caused by less jobs waiting for swapping service and more jobs in memory competing for CPU service.

The following results were obtained from the comparisons of the two class workload model versus the

Analytical Modeling Results

single class workload models (average class and no batch class)

- The no batch workload model seriously underestimates the response time of interactive jobs for all memory sizes, while the average workload model overestimates response times for large main memory sizes and underestimates response times for small main memory sizes.
- The single-class workload models underestimate the CPU and swapper utilizations.
- For all performance measures considered, the average workload model was closer to the two class workload model than the no batch workload model.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

From the results obtained in Chapter 5 for the workload described, the following conclusions can be made

1. Chen's swapping model is a more realistic model of a computer system than the classical swapping model due to the fact that Chen's model more accurately models the effects of the scarcity of main memory on job-swapping behavior.
2. Adding the batch class to Chen's swapping model improves the accuracy of the performance measure predictions when the system contains significant batch activity.

Conclusions and Recommendations

6.2 Recommendations for Future Research

The following are topics of studies which could be conducted as an extension of this thesis

1. The current model of the DECsystem-10 shown in Figure 5-1 is a highly abstract model that does not consider many of the complexities of the TOPS-10 monitor or the hardware configuration. As an extension to the model, the following future thesis efforts could be conducted
 - Modeling the CPU's in the DECsystem-10 as a single processor-sharing node may oversimplify the complex nature of the HPQ's, PQ1, and PQ2 processor queues. Including multiple queues with job class priorities may provide a more accurate model of the overall system behavior. These multiple queues and job class priorities may be implemented using the shortest elapsed time or shortest remaining processing time queueing disciplines (Ref 6:178-186).
 - Modeling the I/O subsystem by just modeling the disk drives may be overlooking long delays due to contention for the I/O channel. An approximation method described in reference 16 called the method of surrogate delays appears to be a prime candidate for inclusion to the present computer program to more accurately model I/O channel contention.
 - An important extension to the present model would be to determine how the numerous scheduling parameters within the DECsystem-10 affect the workload configuration parameters, service times, and the probability transition matrices in the model. This information would provide guidance to systems analysts in setting optimum values for the scheduling parameters.
2. Although the current model does provide reasonable results, the model has not been validated. To accomplish this validation, one of the following two approaches could be taken

Conclusions and Recommendations

- Take measurements on the actual system, and compare the measurement results to the results predicted by the model.
- If measurements on the actual system are impossible, the development of an extremely detailed simulation model which takes into account most of the intricacies of the TOPS-10 monitor would be a step toward validation. This would not completely validate the analytical model since the simulation itself is not validated, but the modeling accuracy possible with a simulation should provide a more accurate, but time consuming model.

3. The impact of changes in the workload configuration using the current model would provide better insights as to how the performance measurements are affected by the workload configuration. Some of the most interesting variations might be changes in the

- Job Size
- Number of batch jobs in the system
- Number of competing jobs in the system, i.e. not in a long-term wait state
- Number of CPU versus I/O intensive jobs.

4. Another type of workload variation entails increasing the number of job classes in the system. To model a large number of job classes (greater than 4) in the system will require a restructuring of the current program. The program currently computes the necessary probability distributions one node at a time, since the state space for the whole network would be too large to run on the CDC computer. With a larger number of job classes, the state space of a node becomes unmanageable. Therefore, the computations will have to be broken down even further, in order to fit in memory. If the program rewrite is undertaken, one of the two alternatives listed below should be chosen

- Rewrite the program in FORTRAN77 to take advantage of the language's increased structure and more importantly, to incorporate the random record reads and writes that are now part of the

Conclusions and Recommendations

language. This will eliminate the installation dependent implementation of random record processing and make the program more portable.

- Rewrite the program in PASCAL, PL/I or some other language that allows recursion. Since the computational algorithms are defined recursively, this may provide more efficient code and will definitely be a more straight forward implementation.

Bibliography

1. Allen, A. O. "Queueing Models of Computer Systems," Computer (April 1980), pp. 13-24.
2. Baskett F., et. al. "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," JACM 22, 2 (April 1975), pp. 598-603.
3. Buzen, J. P. "Computational Algorithms for Closed Queueing Networks with Exponential Servers," Comm. ACM 16, 9 (Sept. 1973), pp. 527-531.
4. Chandy, K. M. and Sauer C. H. "Approximate Methods for Analyzing Queueing Network Models of Computing Systems" ACM Computing Surveys 10, 3 (Sept 1978).
5. Chen, P. P. "Queueing Network Model of Interactive Computing Systems," Proceedings of the IEEE 63, 6 (June 1975), pp. 954-957.
6. Coffman, E. G. and Denning, P. J. Operating Systems Theory, New Jersey: Prentice-Hall, 1973.
7. Control Data Corporation. FORTRAN Extended Version 4 Reference Manual Revision G. Minneapolis, Minnesota: Publication and Graphics Division, Jan. 1981.
8. Digital Equipment Corporation. TOPS-10 Monitor Internals. Revision 6. Maynard, Massachusetts: Digital Equipment Corporation Educational Services, Nov. 1980.
9. Ferrari, D. Computer Systems Performance Evaluation. New Jersey: Prentice-Hall, 1978.
10. Gordon, W. J. and Newell, G. F. "Closed Queueing Systems with Exponential Servers," Operations Research 15, 2 (April 1967), pp. 254-265.
11. Graham, G. S. "Guest Editors Overview: Queueing Network Models of Computer System Performance," ACM Computing Surveys 10, 3 (Sept 1978).
12. Hayes, J. P. Computer Architecture and Organization, New York: McGraw-Hill Book Co., 1978.
13. IMSL Library Reference Manual. Vol. II. IMSL LIB-0008 June 1980.
14. Jackson, J. R. "Networks of Waiting Lines," Operations Research 5 (1957), pp. 518-521.

Bibliography (Continued)

15. "Jobshop-like Queueing Systems," Management Science 10, 1 (Oct 1963), pp. 131-142.
16. Jacobson, P. A. and Lazowska, E. D. "The Method of Surrogate Delays: Simultaneous Resource Possession in Analytical Models of Computer Systems" Performance Evaluation Review 10, 3 (1981), pp. 165-174.
17. Kleinrock, L. Queueing Systems. Volume I: Theory, New York: John Wiley & Sons, Inc., 1975.
18. Madnick, S. E. and Donovan, J. J. Operating Systems, New York: McGraw-Hill Book Co., 1974.
19. McKenzie, L. E. "Queueing Network Model for Performance Evaluation of the DECsystem-10," AFIT-GCS-EE-77-1, Air Force Institute of Technology, 1977.
20. Moore, G. G. Network Models for Large-scale Time-sharing Systems Ph.d. Dissertation. Ann Arbor, Michigan: University of Michigan, 1971.
21. Muntz, R. R. "Analytical Modeling of Interactive Systems," Proceedings of the IEEE 63, 6 (June 1975).
22. "Queueing Networks: A Critique of the State of the Art and Directions for the Future," ACM Computing Surveys 10, 2 (Sept 1978), pp. 353-358.
23. Rose, C. A. "Measurement and Analysis for Computer Performance Evaluation," Ph.d. Dissertation, George Washington University, 1975.
24. Saxton, H. E. "Validation of Closed Queueing Network Models for the DECsystem-10," AFIT-GCS-EE-78-7, Master's Thesis, Air Force Institute Technology, 1978.
25. Sanabria, P. "Design and Verification of Computer Performance Measure Evaluation Analysis Techniques," Ph.d. Dissertation, Texas A&M University, 1977.
26. Shannon, R. E. Systems Simulation: the Art and Science, New Jersey: Prentice-Hall, Inc., 1975.
27. Sloan, M. E. Computer Hardware and Organization: An Introduction, Science Research Associates, Inc., 1976.
28. Smolsky, E., et. al. SCHED: Program Logic Manual for Scheduler and Swapper. Kalamazoo, Michigan: Western Michigan University, 1977.

Bibliography (Continued)

29. Svobodova, L. Computer Performance Measurement and Evaluation Methods: Analysis and Application. Ph.d. Dissertation, Stanford University, 1974.
30. Wong, J. W. Queueing Network Models for Computer Systems. Ph.d. Dissertation UCLA-ENG-7579. Los Angeles, California; UCLA, 1975.

Vita

Michael H. Cox was born in Aberdeen, Maryland on June 20, 1955. Following graduation from Junction City High School in 1973, he attended the USAF Academy, where he received a Bachelor of Science degree in Mathematics in 1977. Prior to this assignment to AFIT, he served as an aircraft vulnerability analyst at the Air Force Flight Dynamics Lab, Wright-Patterson AFB, OH.

Permanent Address:

3139 E. Sahara Avenue, #192
Las Vegas, NV 89104

Appendix A

MULTI-CLASS CHEN MODEL PROGRAM LISTING

PROGRAM PERFORM(INPUT,OUTPUT,TAPES=INPUT,TAPE6=OUTPUT,TAPE3)	PERFORM 2
C	PERFORM 3
C THE PROGRAM PERFORM IS A CLOSED QUEUEING NETWORK	PERFORM 4
C MODEL WHICH CAN BE USED IN PERFORMANCE EVALUATION OF	PERFORM 5
C TIME-SHARING COMPUTER SYSTEMS. THE PROGRAM PERFORM IS	PERFORM 6
C BASED ON A CLOSED QUEUEING NETWORK MODEL DEVELOPED BY	PERFORM 7
C F. BASKETT, K. CHANDY, R. MUNTZ, AND F. PALACIOS.	PERFORM 8
C THEIR MODEL WAS PRESENTED IN THE ARTICLE "OPEN, CLOSED,	PERFORM 9
C AND MIXED NETWORK OF QUEUES WITH DIFFERENT CLASSES OF	PERFORM 10
C CUSTOMERS", WHICH APPEARED IN THE JOURNAL OF THE ACM,	PERFORM 11
C VOL. 22, NO. 2, PP. 248-260, APRIL, 1975.	PERFORM 12
C	PERFORM 13
C THE COMPUTATIONAL ALGORITHMS WHICH ARE USED TO	PERFORM 14
C IMPLEMENT THE MODEL OF BASKETT, ET. AL. ARE BASED ON THE	PERFORM 15
C COMPUTATIONALLY EFFICIENT ALGORITHMS PRESENTED IN THE	PERFORM 16
C DISSERTATION OF J. WONG ENTITLED QUEUEING NETWORK MODELS	PERFORM 17
C FOR COMPUTER SYSTEMS, UCLA-ENG-7579, UNIVERSITY OF	PERFORM 18
C CALIFORNIA AT LOS ANGELES, OCTOBER, 1975.	PERFORM 19
1 DIMENSION E(13,2), EXPVAL(13,2), DEPART(13,2), FNT(1000),	COXMOD 1
1 IFACTOR(41), INDEX3(40), ISKIP(2), ISTATE1(2), ISTATE2(2),	COXMOD 2
2 NODETYP(13), NORMDN(1000,2), NUSERS(2), PROB(13,13),	COXMOD 3
3 PROBCUM(1000), PROBMAR(1000), PROBPAR(13,2,41),	COXMOD 4
4 PROBTOT(13,41), SERVCE(13,2), UTIL(13,2), WAREA(13),	COXMOD 5
5 X(13), IDEP(13), DEP(13,2,40), TCPU(2), DIO(2),	COXMOD 6
6 DISKPRB(13,2), SIZEJOB(2)	COXMOD 7
DOUBLE PRECISION IFACTOR	COXMOD 8
LOGICAL PRINTON	COXMOD 9
INTEGER IUSERS, NAVGJOB	COXMOD 10
REAL DELTA, TOLERAN, PEXPVAL, SIZEMEM, AVGJOBS	COXMOD 11
MAXNODE = 13	COXMOD 12
MAXUSER = 40	COXMOD 13
MAXUSEI = MAXUSER + 1	COXMOD 14
MAXTYPE = 2	COXMOD 15
MAXSTAT = 1000	COXMOD 16
ISIZE3 = 40	COXMOD 17
C MAXNODE — THE MAXIMUM NUMBER OF NODES WHICH CAN BE	PERFORM 33
C REPRESENTED IN THE PROGRAM.	PERFORM 34
C	PERFORM 35
C MAXUSER — THE MAXIMUM NUMBER OF TOTAL USERS THAT CAN	PERFORM 36
C BE REPRESENTED IN THE PROGRAM.	PERFORM 37
C	PERFORM 38
C	PERFORM 39
C MAXUSERI — THE MAXIMUM NUMBER OF TOTAL USERS THAT CAN	PERFORM 40
C BE REPRESENTED IN THE PROGRAM + 1.	PERFORM 41
C	PERFORM 42
C MAXTYPE — THE MAXIMUM NUMBER OF TYPES OF USERS THAT	PERFORM 43
C CAN BE REPRESENTED IN THE PROGRAM.	PERFORM 44
C	PERFORM 45
C MAXSTAT — THE MAXIMUM STATE SPACE FOR EACH NODE.	PERFORM 46
C	PERFORM 47

C ISIZE3 — THE MAXIMUM NUMBER OF RANDOM RECORDS (30)	PERFORM 48
C WHICH CAN BE WRITTEN ON TAPE3 + 1. ISIZE3 IS	PERFORM 49
C CALCULATED BY MAXNODE * 3 + 1.	PERFORM 50
C	PERFORM 51
C E(I,J) — THE MEAN ARRIVAL RATE OF TYPE J USERS AT	PERFORM 52
C SERVICE CENTER I. THE DIMENSIONS OF E ARE	PERFORM 53
C (MAXNODE,MAXTYPE).	PERFORM 54
C	PERFORM 55
C EXPVAL(I,J) — THE EXPECTED NUMBER OF TYPE J USERS	PERFORM 56
C AT NODE I AT EQUILIBRIUM. THE DIMENSIONS	PERFORM 57
C OF EXPVAL ARE (MAXNODE, MAXTYPE).	PERFORM 58
C	PERFORM 59
C DEPART(I,J) — THE DEPARTURE RATE OF TYPE J USERS FROM	PERFORM 60
C NODE I AT EQUILIBRIUM. THE DIMENSIONS	PERFORM 61
C OF DEPART ARE (MAXNODE, MAXTYPE).	PERFORM 62
C	PERFORM 63
C FNT(I) — THE FUNCTION REPRESENTING THE RELATIVE	PERFORM 64
C WEIGHTING OF STATE I IN THE CALCULATIONS OF	PERFORM 65
C THE EQUILIBRIUM STATE PROBABILITY DISTRIBUTIONS.	PERFORM 66
C THE DIMENSION OF FNT IS (MAXSTAT).	PERFORM 67
C	PERFORM 68
C IFACTOR(I) — VALUE OF I FATORIAL. DIMENSION OF	PERFORM 69
C IFACTOR IS (MAXUSER1).	PERFORM 70
C	PERFORM 71
C INDEX3(I) — THE LOCATION OF RECORD I ON THE RANDOM	PERFORM 72
C FILE TAPE3. THE DIMENSION OF INDEX3 IS	PERFORM 73
C (ISIZE3).	PERFORM 74
C	PERFORM 75
C ISKIP — WORKING ARRAY USED TO SKIP OVER UNNEEDED	PERFORM 76
C STATES IN THE STATE SPACE FOR A GIVEN NODE.	PERFORM 77
C THE DIMENSION OF ISKIP IS (MAXTYPE).	PERFORM 78
C	PERFORM 79
C ISTATE1 — WORKING ARRAY USED TO SPECIFY A STATE OF A	PERFORM 80
C GIVEN NODE. THE DIMENSION OF ISTATE1 IS (MAXTYPE).	PERFORM 81
C	PERFORM 82
C ISTATE2 — WORKING ARRAY USED TO SPECIFY A STATE OF A	PERFORM 83
C GIVEN NODE. THE DIMENSION OF ISTATE2 IS	PERFORM 84
C (MAXTYPE).	PERFORM 85
C	PERFORM 86
C NODETYP(I) — SPECIFIES THE TYPE OF NODE I. THE	PERFORM 87
C DIMENSION OF NODETYP IS (MAXNODE).	PERFORM 88
C	PERFORM 89
C NORMCON — WORKING ARRAY USED TO CALCULATE THE NORMAL-	PERFORM 90
C IZATION CONSTANT. THE DIMENSIONS OF NORMCON	PERFORM 91
C ARE (MAXSTAT,2).	PERFORM 92
C	PERFORM 93
C NUSERS(I) — NUMBER OF TYPE I USERS REPRESENTED IN THE	PERFORM 94
C MODEL. THE DIMENSION OF NUSERS IS (MAXTYPE).	PERFORM 95
C	PERFORM 96
C PROB(I,J) — THE PROBABILITY THAT A TASK OF A GIVEN TYPE	PERFORM 97
C WILL PROCEED TO NODE I WHEN IT DEPARTS	PERFORM 98
C NODE J. THE DIMENSIONS OF PROB ARE	PERFORM 99
C (MAXNODE,MAXNODE).	PERFORM 100

```

C                               PERFORM101
C   PROBCUM(I) --- THE PROBABILITY THAT A SUBNETWORK OF      COXMOD 18
C   NODES IN THE NETWORK MODEL IS IN A GIVEN                  COXMOD 19
C   STATE. THE STATES IN THE STATE SPACE ARE                  COXMOD 20
C   INDEXED BY THE SUBSCRIPT I. THE DIMENSION                 COXMOD 21
C   OF PROBCUM IS (MAXSTAT).                                 COXMOD 22
C   PROBMR(I) --- THE PROBABILITY THAT A SINGLE NODE IN      PERFORM108
C   THE NETWORK MODEL IS IN A GIVEN STATE.                   PERFORM109
C   THE STATES IN THE STATE SPACE ARE INDEXED                PERFORM110
C   BY THE SUBSCRIPT I. THE DIMENSION OF                  PERFORM111
C   PROBMR IS (MAXSTAT).                                PERFORM112
C   PROBPAR(I,J,K) --- THE PROBABILITY THAT THERE ARE K      PERFORM113
C   USERS OF TYPE J AT NODE. THE                         PERFORM114
C   DIMENSIONS OF PROBPAR ARE (MAXNODE,                  PERFORM115
C   MAXTYPE, MAXUSE1).                                PERFORM116
C   PROBTOT(I,J) --- THE PROBABILITY THAT THERE ARE J USERS    PERFORM117
C   AT NODE I. THE DIMENSIONS OF PROBTOT                  PERFORM118
C   ARE (MAXNODE,MAXTYPE).                                PERFORM119
C   SERVICE(I,J) --- THE SERVICE TIME OF A TYPE J USER AT     PERFORM120
C   NODE I. THE DIMENSIONS OF SERVICE                   PERFORM121
C   ARE (MAXNODE,MAXTYPE).                                PERFORM122
C   UTIL(I,J) --- THE UTILIZATION OF NODE I BY USERS OF TYPE   PERFORM123
C   J. THE DIMENSIONS OF UTIL ARE (MAXNODE,              PERFORM124
C   MAXTYPE).                                         PERFORM125
C   MAREA --- WORKING ARRAY USED TO CALCULATE THE MEAN ARRIVAL   PERFORM126
C   RATE OF USERS AT THE NODES IN THE NETWORK.           PERFORM127
C   THE DIMENSION OF MAREA IS (MAXNODE).                  PERFORM128
C   X --- WORKING ARRAY USED TO CALCULATE THE MEAN ARRIVAL   PERFORM129
C   RATE OF USERS AT THE NODES IN THE NETWORK.           PERFORM130
C   DIMENSION OF X IS (MAXNODE).                           PERFORM131
C   IDEP(I) --- INDICATES STATE DEPENDENT SERVICE          PERFORM132
C   RATES AT NODE I.                                     PERFORM133
C   1 --- NO STATE DEPENDENT SERVICE RATES               PERFORM134
C   2 --- SERVICE RATES DEPENDENT ON THE TOTAL            PERFORM135
C   NUMBER OF USERS AT THE NODE.                          PERFORM136
C   3 --- SERVICE RATES DEPENDENT ON THE NUMBER            PERFORM137
C   OF USERS OF A TYPE AT THE NODE.                      PERFORM138
C   THE DIMENSION OF IDEP IS (MAXNODE).                  PERFORM139
C   DEP(I,J,K) --- THE RELATIVE SERVICE RATE OF TYPE J USERS   PERFORM140
C   AT NODE I WHEN THERE ARE K USERS PRESENT             PERFORM141
C   AT NODE I. THE DIMENSIONS OF DEP ARE                 PERFORM142
C   (MAXNODE,MAXTYPE,MAXUSER).                            PERFORM143
C   WRITE(6,1)
1   FORMAT(1HL,9X,"THIS IS A CLOSED QUEUEING NETWORK MODEL ",    PERFORM144
1" FOR TIME-SHARING COMPUTER SYSTEMS.")                    PERFORM145
1                                         PERFORM146
1                                         PERFORM147
1                                         PERFORM148
1                                         PERFORM149
1                                         PERFORM150
1                                         PERFORM151
1                                         PERFORM152
1                                         PERFORM153
1                                         PERFORM154

```

```

C OPEN THE MASS STORAGE FILE                                PERFORM155
C CALL OPENMS(3,INDEX3,ISIZE3,0)                           PERFORM156
C CALL INPUT TO INPUT THE DATA NECESSARY TO EXECUTE THE   PERFORM157
C PROGRAM                                                 PERFORM158
C CALL INDATA(PROB, SERVICE, E, NODETYP, NUSERS, MAXNODE,   PERFORM159
1MAXUSER, MAXTYPE, MAXSTAT, NODES, IUSERS, ITYPES, X, INDEX3, ISIZE3,   PERFORM160
2AREA, ISTATES, IDEP, DEP, EXPVAL, NDISKS, MAXITER, TOLERAN,   PERFORM161
3DISKPRB, TCPU, DIO, SIZEJOB, AVGJOBS, NAVGJOB), RETURNS(10)   PERFORM162
PRINTON = .TRUE.                                         COXMOD 23
PEXPVAL = 0.0                                           COXMOD 24
NITER = 0                                              COXMOD 25
DO 4 J=1,ITYRES                                       COXMOD 26
DO 4 I=L,NODES                                         COXMOD 27
IF (J .NE. 1) GO TO 2                                 COXMOD 28
EXPVAL(I,J) = IUSERS/NODES                            COXMOD 29
GO TO 4                                              COXMOD 30
2 EXPVAL(I,J) = 0.0                                    COXMOD 31
4 CONTINUE                                            COXMOD 32
C CALL CHEN75 TO COMPUTE PROBABILITY TRANSITION MATRIX PROB COXMOD 33
C CALL CHEN75(PROB,NDISKS,DISKPRB,TCPU,DIO,NAVGJOB,           COXMOD 34
1EXPVAL,SERVICE,DELTA,PEXPVAL,IUSERS,MAXNODE,MAXTYPE,          COXMOD 35
2ITYPES,NODES,NITER,E,X,AREA)                           COXMOD 36
C CALL FUNCT TO CALCULATE THE FUNCTIONS FNT FOR ALL NODES.    COXMOD 37
C CALL FUNCT(E, SERVICE, IFACTOR, ISTATE1,FNT, NUSERS, NODETYP,   COXMOD 38
1NODES, IUSERS, ITYPES, MAXNODE, MAXUSER, MAXUSE1, MAXTYPE,   COXMOD 39
2MAXSTAT, INDEX3, ISIZE3, ISTATES, IDEP, DEP), RETURNS(10)     COXMOD 40
C CALL NORMAL TO CALCULATE THE NORMALIZATION CONSTANT.        COXMOD 41
C CALL NORMAL(FNT, NORMCON, NUSERS, ISTATE1, ISTATE2,ISKIP,   COXMOD 42
1MAXNODE, MAXSTAT, NODES, ITYPES, MAXTYPE, INDEX3, ISIZE3, ISTATES) PERFORM167
C CALL MARGIN TO CALCULATE THE MARGINAL PROBABILITY          PERFORM168
C DISTRIBUTION FOR THE STATE OF THE NODES.                  PERFORM171
C CALL MARGIN(FNT, NORMCON, PROBMAR, PROBCUM, NUSERS, ISTATE1, ISTATE2,   PERFORM172
1ISKIP, MAXNODE, MAXSTAT, MAXTYPE, NODES, ITYPES, INDEX3, ISIZE3, ISTATES) PERFORM173
C CALL EXPECT TO CALCULATE VARIOUS PERFORMANCE MEASURES.    PERFORM174
C CALL EXPECT(PROBMAR,PROBTOT,PROBPAR,NUSERS,ISTATE1,         PERFORM175
1NODETYP,EXPVAL,DEPART,UTIL,SERVICE,E,MAXNODE,               PERFORM176
2MAXUSE1,MAXTYPE,NODES,ITYPES,IUSERS,MAXSTAT,INDEX3,ISIZE3,ISTATES,   PERFORM177
3IDEP,DEP,MAXUSER,PRINTON,NITER)                          PERFORM178
DO 7 L = 1, NODES                                         PERFORM179

```

DO 7 M = 2, ITYPES	COXMOD 45
EXPVAL(L,1) = EXPVAL(L,1) + EXPVAL(L,M)	COXMOD 46
7 CONTINUE	COXMOD 47
DELTA = ABS(PEXPVAL-EXPVAL(NDISKS+3,1))	COXMOD 48
PEXPVAL = EXPVAL(NDISKS+3,1)	COXMOD 49
NITER = NITER + 1	COXMOD 50
PRINTON = .FALSE.	COXMOD 51
WRITE(6,1210) DELTA	COXMOD 52
1210 FORMAT(//10X,"DELTA = ",F10.5)	COXMOD 53
IF ((NITER .LT. MAXITER) .AND. (DELTA .GT. TOLERAN)) GO TO 5	COXMOD 54
PRINTON = .TRUE.	COXMOD 55
CALL EXPECT(PROBMR,PROBTOT,PROBPAR,NUSERS,ISTATE1,	COXMOD 56
1NODETYP,EXPVAL,DEPART,UTIL,SERVICE,E,MAXNODE,	COXMOD 57
2MAXUSEL,MAKTYPE,NODES,ITYPES,IUSERS,MAXSTAT,INDEX3,ISIZE3,ISTATES,	COXMOD 58
3IDEP,DEP,MAXUSER,PRINTON,NITER)	COXMOD 59
GO TO 20	COXMOD 60
10 WRITE(6,15)	PERFORM191
15 FORMAT(//10X,"ERRORS HAVE OCCURRED IN THE INPUT DESCRIPTION.",	PERFORM192
1//2X,"CHECK YOUR INPUT DATA FOR CORRECTNESS.")	PERFORM193
20 CONTINUE	PERFORM194
C CLOSE THE MASS STORAGE FILE.	PERFORM195
C CALL CLOSM3(3)	PERFORM196
STOP	PERFORM197
END	PERFORM198
SUBROUTINE INDATA(PROB,SERVICE,E,1NODETYP,NUSERS,MAXNODE,	INDATA 2
1MAXUSER,MAKTYPE,MAXSTAT,NODES,IUSERS,ITYPES,X,INDEX3,ISIZE3,	INDATA 3
2WAREA,ISTATES,1DEP,DEP,EXPVAL,NDISKS,MAXITER,TOLERAN,	COXMOD 61
3DISKPRB,TCPU,DIO,SIZEJOB,AvgJOBS,NAvgJOBS),RETURNS(ERROR)	COXMOD 62
DIMENSION PROB(MAXNODE,MAXNODE), SERVICE(MAXNODE,MAKTYPE),	INDATA 5
E(MAXNODE,MAKTYPE), NODETYP(MAXNODE), NUSERS(MAKTYPE),	INDATA 6
2X(MAXNODE), WAREA(MAXNODE), INDEX3(ISIZE3), 1DEP(MAXNODE),	INDATA 7
3DEP(MAXNODE,MAKTYPE,MAXUSER),EXPVAL(MAXNODE,MAKTYPE),	COXMOD 63
4DISKPRB(MAXNODE,MAKTYPE),TCPU(MAKTYPE),DIO(MAKTYPE),	COXMOD 64
5SIZEJOB(MAKTYPE)	COXMOD 65
C SUBROUTINE INPUT READS THE DATA NECESSARY TO RUN THE	INDATA 9
C PROGRAM. THE CHARACTERISTICS OF THE TIME-SHARING COMPUTER	INDATA 10
C SYSTEM BEING MODELED ARE INPUT IN THIS ROUTINE.	INDATA 11
C	INDATA 12
WRITE(6,1)	INDATA 13
I FORMAT(//10X,"*****THE INPUT TO THE MODEL FOLLOWS*****")	INDATA 14
C READ THE NUMBER OF NODES IN THE NETWORK AND THE NUMBER	INDATA 15
C OF TYPES OF TASKS.	INDATA 16
C	INDATA 17
READ(5,*1) NODES, ITYPES	INDATA 18
IF (EOF(5)) 1000,15	INDATA 19
15 CONTINUE	COXMOD 66
IF ((NODES.GT. MAXNODE) .OR. (ITYPES.GT. MAKTYPE)) GO TO 1100	INDATA 22
WRITE(6,20) NODES, ITYPES	INDATA 23
20 FORMAT(//10X,"THE NUMBER OF NODES REPRESENTED IN THE MODEL = ",	INDATA 24
	INDATA 25
	INDATA 26

```

1I3//10X,"THE NUMBER OF TASK TYPES REPRESENTED IN THE ",           INDATA 27
2"MODEL = ",I3)                                                 INDATA 28
READ(5,*) NDISKS,SIZEMEM,MAXITER,TOLERAN                         COXMOD 67
WRITE(6,26) NDISKS, SIZEMEM                                         COXMOD 68
26 FORMAT(//10X,"THE NUMBER OF DISKS REPRESENTED IN THE MODEL = ",   COXMOD 69
1I3//10X,"THE SIZE OF MEMORY IN K BYTES = ",F10.5)                 COXMOD 70
IF ( EOF(5) .EQ. 1 ) GO TO 1000                                     COXMOD 71
WRITE(6,27) MAXITER, TOLERAN                                       COXMOD 72
27 FORMAT(//10X,"THE MAXIMUM NUMBER OF ITERATIONS = ",I3//10X,      COXMOD 73
1"THE TOLERANCE FOR CONVERSION = ",F10.5)                           COXMOD 74
IF ( EOF(5) .EQ. 1 ) GO TO 1000                                     COXMOD 75
DO 29 I = 1, NDISKS                                              COXMOD 76
READ(5,*) (DISKPRB(I,J),J=1,ITYPES)                                COXMOD 77
IF (EOF(5) .EQ. 1) GO TO 1000                                      COXMOD 78
29 CONTINUE
WRITE(6,30)
30 FORMAT(//10X,"DISK #", 15X,"DISK ACCESS PROBABILITIES "        COXMOD 81
1"BY TASK TYPE")                                                 COXMOD 82
DO 32 I = 1, NDISKS                                              COXMOD 83
WRITE(6,31) I,(DISKPRB(I,J),J=1,ITYPES)                            COXMOD 84
31 FORMAT(//10X,I4,10X,10(F10.5))                                    COXMOD 85
32 CONTINUE
READ(5,*) (NUSERS(J),TCPUI(J),DIO(J),SIZEJ(O(J)), J=1,ITYPES)    COXMOD 86
IUSERS = 0
ISTATES = 1
WRITE(6,35)
35 FORMAT(///10X,"THE DISTRIBUTION OF TASKS IN THE SYSTEM ",       COXMOD 88
1"BY TASK TYPE //17X,"TASK TYPE",8X,"NUMBER OF TASKS",8X,          COXMOD 89
2"CPU TIME/INTERACTION",8X,"DISK IO'S/INTERACTION",8X,              COXMOD 90
3"JOB SIZE IN K BYTES//")                                         COXMOD 91
DO 41 I = 1, ITYPES
WRITE(6,40) I, NUSERS(I), TCPUI(I), DIO(I), SIZEJOB(I)            COXMOD 92
40 FORMAT(/20X,I3,20X,I3,13X,F10.5,2(20X,F10.5))
IUSERS = IUSERS + NUSERS(I)
ISTATES = ISTATES + (NUSERS(I) + 1)
41 CONTINUE
WRITE(6,42) IUSERS
42 FORMAT(//17X,"TOTAL",21X,I3)                                     COXMOD 97
AVGJOBS = 0.0
DO 43 I = 1, ITYPES
AVGJOBS = AVGJOBS + SIZEJOB(I) + NUSERS(I)                        COXMOD 98
43 CONTINUE
AVGJOBS = AVGJOBS / IUSERS
NAVJOB = SIZEMEM / AVGJOBS                                         COXMOD 99
WRITE(6,44) AVGJOBS,NAVJOB                                         COXMOD 100
44 FORMAT(//10X,"AVERAGE JOB SIZE = ",F10.5,/10X,
1"AVERAGE NUMBER OF JOBS IN MEMORY = ",I3)                         COXMOD 101
IF ( (IUSERS .GT. MAXUSER) .OR. (ISTATES .GT. MAXSTAT) ) GO TO 1200
C
C READ THE NODE CHARACTERISTICS.
C
48 WRITE(6,48) ITYPES
48 FORMAT(///10X,"NODE CHARACTERISTICS"/10X,"NODE",5X,"NODE",5X,    INDATA 46
INDATA 47
INDATA 48
INDATA 49
INDATA 50
INDATA 51

```

```

1"SERVICE"/10X,"NAME",5X,"TYPE",7X,"RATE",6X,"NODE SERVICE RATES ",      INDATA 52
2"FOR TASK TYPES 1 THROUGH",I3)                                         INDATA 53
  DO 65 I = 1, NODES
    READ(5,I) NODETYP(I), IDEP(I), (SERVICE(I,J),J=1,ITYPES)
    IF( EOF(5) ) 1000,55
55  CONTINUE
    IF ( IDEP(I) .EQ. 2) READ(5,*) (DEP(I,1,J),J=1,MAXUSER)
    IF ( IDEP(I) .NE. 3) GO TO 59
    DO 58 J = 1, ITYPES
      READ(5,*) (DEP(I,J,K),K=1,MAXUSER)
58  CONTINUE
59  CONTINUE
    WRITE(6,60) I, NODETYP(I), IDEP(I), (SERVICE(I,J),J=1,ITYPES)          INDATA 64
60  FORMAT(/10X,I4.5X,I4.8X,I4.6X,8F10.5,10I4X,8F10.5))                  INDATA 65
65  CONTINUE
    RETURN
1000 WRITE(6,1005)
1005 FORMAT(/10X,"***UNEXPECTED END OF INPUT***")
    RETURN ERROR
1100 WRITE(6,1105) MAXNODE, MAXTYPE, NODES, ITYPES                      INDATA 126
1105 FORMAT(/10X,"MAXIMUM NODES = ",I3,"MAXIMUM TASK TYPES = ",I3//,        INDATA 127
  1"YOUR INPUTS ARE ",I3," AND ",I3," RESPECTIVELY.")
    RETURN ERROR
1200 WRITE(6,1205) MAXUSER, MAXSTAT, IUSERS, ISTATES                     INDATA 129
1205 FORMAT(/10X,"MAXIMUM NUMBER OF TASKS = ",I3,/10X,                      INDATA 130
  1"MAXIMUM NUMBER OF STATES = ",I5,/10X,
  2"YOUR INPUTS ARE ",I3," AND ",I5," RESPECTIVELY.")
    RETURN ERROR
    END
    SUBROUTINE FUNCT(E,SERVICE,IFACTOR,ISTATE,FNT,NUSERS,NODETYP,
  1NODES,IUSERS,ITYPES,MAXNODE,MAXUSER,MAXUSE1,
  2MAXTYPE,MAXSTAT,INDEX3,ISIZE3,ISTATES,IDEPEP,DEP),RETURNS(ERROR)
    DIMENSION E(MAXNODE,MAXTYPE), SERVICE(MAXNODE,MAXTYPE),
  1IFACTOR(MAXUSE1), ISTATE(MAXTYPE), FNT(MAXSTAT),
  2NUSERS(MAXTYPE), NODETYP(MAXNODE), INDEX3(ISIZE3),
  3IDEPEP(MAXNODE), DEP(MAXNODE,MAXTYPE,MAXUSER)
    DOUBLE PRECISION IFACTOR
C
C      THIS SUBROUTINE CALCULATES THE VALUES FOR THE FUNCTIONS
C      WHICH ARE USED TO CALCULATE THE PROBABILITY THAT THE COMPUTER
C      SYSTEM BEGINS MODELED IS IN A GIVEN STATE AT EQUILIBRIUM.
C
  IFACTOR(1) = 1
  DO 10 I = 1, MAXUSER
    IFACTOR(I+1) = IFACTOR(I) * I
10  CONTINUE
  DO 60 I = 1, NODES
    INDEX = 1
    DO 20 J = 1, ITYPES
      ISTATE(J) = 0
20  CONTINUE
25  CONTINUE
    NUMBER = 1

```

	INDATA	52
	INDATA	53
	INDATA	54
	CXMOD	107
	INDATA	57
	INDATA	58
	CXMOD	108
	INDATA	61
	INDATA	62
	CXMOD	109
	INDATA	64
	INDATA	65
	INDATA	66
	INDATA	67
	INDATA	68
	INDATA	122
	INDATA	123
	INDATA	124
	INDATA	125
	INDATA	126
	INDATA	127
	INDATA	128
	INDATA	129
	INDATA	130
	CXMOD	110
	CXMOD	111
	CXMOD	112
	INDATA	133
	INDATA	134
	FUNCT	2
	FUNCT	3
	FUNCT	4
	FUNCT	5
	FUNCT	6
	FUNCT	7
	FUNCT	8
	FUNCT	9
	FUNCT	10
	FUNCT	11
	FUNCT	12
	FUNCT	13
	FUNCT	14
	FUNCT	15
	FUNCT	16
	FUNCT	17
	FUNCT	18
	FUNCT	19
	FUNCT	20
	FUNCT	21
	FUNCT	22
	FUNCT	23
	FUNCT	24
	FUNCT	25

	DO 30 J = 1, ITYPES	FUNCTION	26
30	NUMBER = NUMBER + ISTATE(J)	FUNCTION	27
	CONTINUE	FUNCTION	28
	J = 2	FUNCTION	29
35	CONTINUE	FUNCTION	30
	FVALUE1 = 1.	FUNCTION	31
	FVALUE2 = 1.	FUNCTION	32
	DO 38 K = 1, ITYPES	FUNCTION	33
	L = ISTATE(K)	FUNCTION	34
	FVALUE1 = FVALUE1 * ((E(I,K) * SERVICE(I,K))**L)	FUNCTION	35
	FVALUE2 = FVALUE2 * IFACTOR(L+1)	FUNCTION	36
38	CONTINUE	FUNCTION	37
	IF (NODETYP(I) .EQ. 3) FVALUE2 = 1. / FVALUE2	FUNCTION	38
	IF (NODETYP(I) .NE. 3) FVALUE2 = IFACTOR(NUMBER) / FVALUE2	FUNCTION	39
	FNT(INDEX) = FVALUE1 * FVALUE2	FUNCTION	40
	IF (IDEP(I) .NE. 2) GO TO 41	FUNCTION	41
	NUM = NUMBER - 1	FUNCTION	42
	IF (NUM .EQ. 0) GO TO 41	FUNCTION	43
	FFF = 1.	FUNCTION	44
	DO 40 K = 1, NUM	FUNCTION	45
	FFF = FFF * DEP(I,1,K)	FUNCTION	46
40	CONTINUE	FUNCTION	47
	FNT(INDEX) = FNT(INDEX) / FFF	FUNCTION	48
41	CONTINUE	FUNCTION	49
	IF (IDEP(I) .NE. 3) GO TO 44	FUNCTION	50
	FFF = 1.	FUNCTION	51
	DO 43 K = 1, ITYPES	FUNCTION	52
	IF (ISTATE(K) .EQ. 0) GO TO 43	FUNCTION	53
	NUM = ISTATE(K)	FUNCTION	54
	DO 42 KK = 1, NUM	FUNCTION	55
	FFF = FFF * DEP(I,K,KK)	FUNCTION	56
42	CONTINUE	FUNCTION	57
43	CONTINUE	FUNCTION	58
	FNT(INDEX) = FNT(INDEX) / FFF	FUNCTION	59
44	CONTINUE	FUNCTION	60
	INDEX = INDEX + 1	FUNCTION	61
	NUMBER = NUMBER + 1	FUNCTION	62
	ISTATE(1) = ISTATE(1) + 1	FUNCTION	63
	IF (ISTATE(1) .LE. NUSERS(1)) GO TO 35	FUNCTION	64
45	CONTINUE	FUNCTION	65
	ISTATE(J) = ISTATE(J) + 1	FUNCTION	66
	IF (ISTATE(J) .GT. NUSERS(J)) GO TO 55	FUNCTION	67
	K = J - 1	FUNCTION	68
	DO 50 L = 1,K	FUNCTION	69
	ISTATE(L) = 0	FUNCTION	70
50	CONTINUE	FUNCTION	71
	GO TO 25	FUNCTION	72
55	CONTINUE	FUNCTION	73
	J = J + 1	FUNCTION	74
	IF (J .LE. ITYPES) GO TO 45	FUNCTION	75
C		FUNCTION	76
C	THE FUNCTIONAL VALUES CALCULATED FOR ALL POSSIBLE STATES	FUNCTION	77
C	OF EACH NODE ARE SAVED SO THAT THEY CAN BE USED TO CALCULATE	FUNCTION	78

C	THE NORMALIZATION CONSTANT.	FUNCT	79
C		FUNCT	80
C	CALL WRITNS(3,FNT(1),ISTATES,I)	FUNCT	81
60	CONTINUE	FUNCT	82
	RETURN	FUNCT	83
	END	FUNCT	84
	SUBROUTINE NORMAL(FNT,NORMCON,NUSERS,ISTATE1,	FUNCT	85
1	ISTATE2,ISKIP,MAXNODE,MAXSTAT,NODES,ITYPES,	NORMAL	2
2	MAXTYPE,INDEX3,ISIZE3,ISTATES)	NORMAL	3
DIMENSION FNT(MAXSTAT), NORMCON(MAXSTAT,2),	NORMAL	4	
1	NUSERS(MAXTYPE), ISTATE1(MAXTYPE), ISTATE2(MAXTYPE),	NORMAL	5
2	ISKIP(MAXTYPE),INDEX3(ISIZE3)	NORMAL	6
INTEGER FINDEX	NORMAL	7	
REAL NORMCON	NORMAL	8	
C		NORMAL	9
C	THIS SUBROUTINE CALCULATES THE NORMALIZATION CONSTANT	NORMAL	10
C	FOR THE MODEL. THE NORMALIZATION CONSTANT ASSURES THAT THE	NORMAL	11
C	PROBABILITY OF THE SYSTEM BEING IN ALL STATES SUMS TO UNITY.	NORMAL	12
C		NORMAL	13
	CALL READNS(3,NORMCON(1,1),ISTATES,1)	NORMAL	14
	CALL WRITNS(3,NORMCON(1,1),ISTATES,MAXNODE+1)	NORMAL	15
	NORM1 = 1	NORMAL	16
	NORM2 = 2	NORMAL	17
DO 110 I = 2, NODES	NORMAL	18	
	CALL READNS(3,FNT(1),ISTATES,I)	NORMAL	19
	INDEX = 1	NORMAL	20
DO 15 J = 1, ITYPES	NORMAL	21	
	ISTATE1(J) = 0	NORMAL	22
15	CONTINUE	NORMAL	23
20	CONTINUE	NORMAL	24
K = 2	NORMAL	25	
DO 25 J = 1, ITYPES	NORMAL	26	
	ISTATE2(J) = 0	NORMAL	27
25	CONTINUE	NORMAL	28
KK = 1	NORMAL	29	
FINDEX = 1	NORMAL	30	
LINDEX = INDEX	NORMAL	31	
ITEMP = 1	NORMAL	32	
DO 30 J = 1, ITYPES	NORMAL	33	
	ISKIP(J) = ITEMp + (NUSERS(J) - ISTATE1(J))	NORMAL	34
	ITEMP = ITEMp + (NUSERS(J) + 1)	NORMAL	35
30	CONTINUE	NORMAL	36
	ISKIP(1) = ISKIP(1) + 1	NORMAL	37
	FVALUE = 0.	NORMAL	38
40	CONTINUE	NORMAL	39
	FVALUE = FVALUE + NORMCON(FINDEX,NORM1) + FNT(LINDEX)	NORMAL	40
	IF (FINDEX .EQ. INDEX) GO TO 70	NORMAL	41
	ISTATE2(KK) = ISTATE2(KK) + 1	NORMAL	42
	IF (ISTATE2(KK) .GT. ISTATE1(KK)) GO TO 50	NORMAL	43
	FINDEX = FINDEX + 1	NORMAL	44
	LINDEX = LINDEX - 1	NORMAL	45
	GO TO 40	NORMAL	46
		NORMAL	47

```

50  CONTINUE          NORMAL  48
      FINDEX = FINDEX + ISKIP(KK)
      LINDEX = LINDEX - ISKIP(KK)
      KK = KK + 1
      ISTATE2(KK) = ISTATE2(KK) + 1
      IF ( ISTATE2(KK) .GT. ISTATE1(KK) ) GO TO 50
      L = KK - 1
      DO 60 J = 1, L
      ISTATE2(J) = 0
60  CONTINUE          NORMAL  49
      KK = 1
      GO TO 40
70  CONTINUE          NORMAL  50
      NORMCON(INDEX,NORM2) = FVALUE
      INDEX = INDEX + 1
      ISTATE1(1) = ISTATE1(1) + 1
      IF ( ISTATE1(1) .LE. NUSERS(1) ) GO TO 20
80  CONTINUE          NORMAL  51
      ISTATE1(K) = ISTATE1(K) + 1
      IF ( ISTATE1(K) .GT. NUSERS(K) ) GO TO 100
      L = K - 1
      DO 90 J= 1, L
      ISTATE1(J) = 0
90  CONTINUE          NORMAL  52
      GO TO 20
100 CONTINUE          NORMAL  53
      K = K + 1
      IF ( K .LE. ITYPES ) GO TO 80
      CALL WRTHS(3,NORMCON(1,NORM2),ISTATES,MAXNODE+I)
      ITEMP = NORM1
      NORM1 = NORM2
      NORM2 = ITEMP
110 CONTINUE          NORMAL  54
      RETURN
      END
      SUBROUTINE MARGIN(FNT,CONNORM,PROBMAR,PROBCUM,
     1NUSERS,ISTATE1,ISTATE2,ISKIP,MAXNODE,MAXSTAT,
     2MAXTYPE,NODES,ITYPES,INDEX3,ISIZE3,ISTATES)
      DIMENSION FNT(MAXSTAT),CONNORM(MAXSTAT,2),
     1PROBMAR(MAXSTAT),PROBCUM(MAXSTAT),
     2NUSERS(MAXTYPE),ISTATE1(MAXTYPE),ISTATE2(MAXTYPE),
     3ISKIP(MAXTYPE),INDEX3(ISIZE3)
      INTEGER FINDEX
C
C      THIS SUBROUTINE CALCULATES THE PROBABILITY THAT A GIVEN
C      NODE IS IN A GIVEN STATE.  THIS PROBABILITY IS CALCULATED
C      FOR ALL POSSIBLE STATES FOR EACH NODE.
C
      NORM1 = 1
      NORM2 = 2
      CALL READHS(3,CONNORM(1,NORM1),ISTATES,MAXNODE+NODES)
      CONSTAN = 1. / CONNORM(ISTATES,NORM1)
      FINDEX = 1
      NORMAL  55
      NORMAL  56
      NORMAL  57
      NORMAL  58
      NORMAL  59
      NORMAL  60
      NORMAL  61
      NORMAL  62
      NORMAL  63
      NORMAL  64
      NORMAL  65
      NORMAL  66
      NORMAL  67
      NORMAL  68
      NORMAL  69
      NORMAL  70
      NORMAL  71
      NORMAL  72
      NORMAL  73
      NORMAL  74
      NORMAL  75
      NORMAL  76
      NORMAL  77
      NORMAL  78
      NORMAL  79
      NORMAL  80
      NORMAL  81
      NORMAL  82
      MARGIN  2
      MARGIN  3
      MARGIN  4
      MARGIN  5
      MARGIN  6
      MARGIN  7
      MARGIN  8
      MARGIN  9
      MARGIN 10
      MARGIN 11
      MARGIN 12
      MARGIN 13
      MARGIN 14
      MARGIN 15
      MARGIN 16
      MARGIN 17
      MARGIN 18
      MARGIN 19

```

LINDEX = ISTATES	MARGIN 20
PPP = 0.	MARGIN 21
CALL READMS(3,FNT(1),ISTATES,NODES)	MARGIN 22
CALL READMS(3,CONNORM(1,NORM2),ISTATES,MAXNODE+NODES-1)	MARGIN 23
20 CONTINUE	MARGIN 24
PROBMAR(FINDEX) = CONSTAN * CONNORM(LINDEX,NORM2) *	MARGIN 25
1FNT(FINDEX)	MARGIN 26
PROBCUM(FINDEX) = CONSTAN * CONNORM(FINDEX,NORM2) *	MARGIN 27
1FNT(LINDEX)	MARGIN 28
PPP = PPP + PROBMAR(FINDEX)	MARGIN 29
FINDEX = FINDEX + 1	MARGIN 30
LINDEX = LINDEX - 1	MARGIN 31
IF (FINDEX .LE. ISTATES) GO TO 20	MARGIN 32
IF (NODES .LE. 2) GO TO 150	MARGIN 33
NCYCLES = NODES - 2	MARGIN 34
CALL WRITMS(3,PROBMAR(1),ISTATES,MAXNODE*2+NODES)	MARGIN 35
DO 140 I = 1, NCYCLES	MARGIN 36
NSUB = NODES - I	MARGIN 37
CALL READMS(3,FNT(1),ISTATES,NSUB)	MARGIN 38
ITEMP = NORM1	MARGIN 39
NORM1 = NORM2	MARGIN 40
NORM2 = ITEMP	MARGIN 41
CALL READMS(3,CONNORM(1,NORM2),ISTATES,MAXNODE+NSUB-1)	MARGIN 42
INDEX = 1	MARGIN 43
DO 30 J = 1, ITYPES	MARGIN 44
ISTATE1(J) = 0	MARGIN 45
30 CONTINUE	MARGIN 46
K = 2	MARGIN 47
PPP = 0.	MARGIN 48
40 CONTINUE	MARGIN 49
DO 50 J = 1, ITYPES	MARGIN 50
ISTATE2(J) = ISTATE1(J)	MARGIN 51
50 CONTINUE	MARGIN 52
FINDEX = 1	MARGIN 53
LINDEX = INDEX	MARGIN 54
KK = 1	MARGIN 55
ITEMP = 1	MARGIN 56
DO 60 J = 1, ITYPES	MARGIN 57
ISKIP(J) = ITEMP * ISTATE1(J)	MARGIN 58
ITEMP = ITEMP + (NUERS(J) + 1)	MARGIN 59
60 CONTINUE	MARGIN 60
ISKIP(1) = ISKIP(1) + 1	MARGIN 61
FVALUE1 = 0.	MARGIN 62
FVALUE2 = 0.	MARGIN 63
70 CONTINUE	MARGIN 64
FVALUE1 = FVALUE1 + PROBCUM(LINDEX) *	MARGIN 65
1((CONNORM(FINDEX,NORM2) * FNT(INDEX)) /	MARGIN 66
2CONNORM(LINDEX,NORM1))	MARGIN 67
FVALUE2 = FVALUE2 + PROBCUM(LINDEX) *	MARGIN 68
1((CONNORM(INDEX,NORM2) * FNT(FINDEX)) /	MARGIN 69
2CONNORM(LINDEX,NORM1))	MARGIN 70
IF (LINDEX .EQ. ISTATES) GO TO 100	MARGIN 71
ISTATE2(KK) = ISTATE2(KK) + 1	MARGIN 72

```

IF ( ISTATE2(KK) .GT. NUSERS(KK) ) GO TO 80          MARGIN 73
FINDEX = FINDEX + 1                                 MARGIN 74
LINDEX = LINDEX + 1                                 MARGIN 75
GO TO 70                                         MARGIN 76
80  CONTINUE
FINDEX = FINDEX + ISKIP(KK)                         MARGIN 77
LINDEX = LINDEX + ISKIP(KK)                         MARGIN 78
KK = KK + 1                                         MARGIN 79
ISTATE2(KK) = ISTATE2(KK) + 1                        MARGIN 80
IF (ISTATE2(KK) .GT. NUSERS(KK) ) GO TO 80          MARGIN 81
L = KK - 1                                         MARGIN 82
DO 90 J = 1, L                                     MARGIN 83
ISTATE2(J) = ISTATE1(J)                            MARGIN 84
90  CONTINUE
KK = 1                                              MARGIN 85
GO TO 70                                         MARGIN 86
100 CONTINUE
PROBMAR(INDEX) = FVALUE1                           MARGIN 87
PPP = PPP + FVALUE1                               MARGIN 88
PROBCUM(INDEX) = FVALUE2                           MARGIN 89
INDEX = INDEX + 1                                 MARGIN 90
ISTATE1(1) = ISTATE1(1) + 1                        MARGIN 91
IF (ISTATE1(1) .LE. NUSERS(1) ) GO TO 40          MARGIN 92
110  CONTINUE
ISTATE1(K) = ISTATE1(K) + 1                        MARGIN 93
IF (ISTATE1(K) .GT. NUSERS(K) ) GO TO 130          MARGIN 94
L = K - 1                                         MARGIN 95
DO 120 J = L, L                                   MARGIN 96
ISTATE1(J) = 0                                     MARGIN 97
120  CONTINUE
K = 2                                              MARGIN 98
GO TO 40                                         MARGIN 99
130  CONTINUE
K = K + 1                                         MARGIN 100
IF ( K .LE. ITYPES ) GO TO 110                     MARGIN 101
CALL WRITMS(3,PROBMAR(),ISTATES,MAXNODE+2+NSUB)   MARGIN 102
140  CONTINUE
150  CONTINUE
PPP = 0.                                            MARGIN 103
DO 160 I = 1, ISTATES                           MARGIN 104
PPP = PPP + PROBCUM(I)                           MARGIN 105
160  CONTINUE
CALL WRITMS(3,PROBCUM(),ISTATES,MAXNODE+2+1)      MARGIN 106
RETURN
END
SUBROUTINE EXPECT(PROBMAR,PROBTOT,PROBPAR,NUSERS,ISTATE,
1NODETYP,EXPVAL,DEPART,UTIL,SERVICE,E,
2MAXNODE,MAXUSE1,MAXTYPE,NODES,ITYPES,IUSERS,MAXSTAT,INDEX3,
3ISIZE3,ISTATES,1DEP,DEP,MAXUSER,PRINTN,NITER)
DIMENSION PROBMAR(MAXSTAT),PROBTOT(MAXNODE,MAXUSE1),
1PROBPAR(MAXNODE,MAXTYPE,MAXUSE1), NUSERS(MAXTYPE),
2ISTATE(MAXTYPE), NODETYP(MAXNODE), EXPVAL(MAXNODE,MAXTYPE),
3DEPART(MAXNODE,MAXTYPE), UTIL(MAXNODE,MAXTYPE),
EXPECT 2
EXPECT 3
EXPECT 4
COXMOD 113
EXPECT 6
EXPECT 7
EXPECT 8
EXPECT 9

```

```

4$ERVICE(MAXNODE,MAXTYPE), INDEX3(I$IZE3), E(MAXNODE,MAXTYPE),
5IDEF(MAXNODE), DEP(MAXNODE,MAXTYPE,MAXUSER)
LOGICAL PRINTON
C
C      THIS SUBROUTINE CALCULATES VARIOUS PERFORMANCE MEASURES
C      FOR THE TIME-SHARING SYSTEM BEING MODELED. THESE MEASURES
C      INCLUDE THE MARGINAL QUEUE LENGTH DISTRIBUTION FOR EACH TYPE
C      OF USER AT EACH NODE, THE UTILIZATION OF THE NODES BY EACH
C      TYPE OF USER, AND THE RESPONSE TIME FOR EACH TYPE OF USER.
C
DO 10 I = 1, MAXNODE
DO 10 J = 1, MAXUSEL
PROBTOT(I,J) = 0.0
DO 10 K = 1, MAXTYPE
PROBPAR(I,K,J) = 0.0
UTIL(I,K) = 0.0
DEPART(I,K) = 0.0
10 CONTINUE
IF (PRINTON) WRITE(6,1)
1 FORMAT(1H1,///10X,"*****THE OUTPUT OF THE MODEL FOLLOWS*****")
DO 70 I = 1, NODES
CALL READNS(3,PROBMAR(1),ISTATES,MAXNODE*2+I)
K = 1
INDEX = 0
NTOTAL = 0
DO 20 J = 1, ITYPES
ISTATE(J) = 1
20 CONTINUE
40 CONTINUE
NTOTAL = NTOTAL + 1
INDEX = INDEX + 1
PROB = PROBMAR(INDEX)
PROBTOT(I,NTOTAL) = PROBTOT(I,NTOTAL) + PROB
DO 50 J = 1, ITYPES
JJ = ISTATE(J)
PROBPAR(I,J,JJ) = PROBPAR(I,J,JJ) + PROB
IF ( JJ .EQ. 1 ) GO TO 50
PARTIAL = PROB + (JJ - 1)
IF ( NODETYP(I) .EQ. 3 ) GO TO 45
PARTIAL = PARTIAL / (NTOTAL - 1)
UTIL(I,J) = UTIL(I,J) + PARTIAL
45 CONTINUE
PARTIAL = PARTIAL + (1. / SERVICE(I,J))
DEPART(I,J) = DEPART(I,J) + PARTIAL
50 CONTINUE
IF ( INDEX .EQ. ISTATES ) GO TO 70
60 CONTINUE
ISTATE(K) = ISTATE(K) + 1
IF ( ISTATE(K) .GT. NUSERS(K) + 1 ) GO TO 65
K = 1
GO TO 40
65 CONTINUE
ISTATE(K) = 1

```

EXPECT 10
 EXPECT 11
 COXMOD 114
 EXPECT 12
 EXPECT 13
 EXPECT 14
 EXPECT 15
 EXPECT 16
 EXPECT 17
 EXPECT 18
 COXMOD 115
 COXMOD 116
 COXMOD 117
 COXMOD 118
 COXMOD 119
 COXMOD 120
 COXMOD 121
 COXMOD 122
 COXMOD 123
 COXMOD 124
 EXPECT 21
 EXPECT 22
 EXPECT 23
 EXPECT 24
 EXPECT 25
 EXPECT 26
 EXPECT 27
 EXPECT 28
 EXPECT 29
 EXPECT 30
 EXPECT 31
 EXPECT 32
 EXPECT 33
 EXPECT 34
 EXPECT 35
 EXPECT 36
 EXPECT 37
 EXPECT 38
 EXPECT 39
 EXPECT 40
 EXPECT 41
 EXPECT 42
 EXPECT 43
 EXPECT 44
 EXPECT 45
 EXPECT 46
 EXPECT 47
 EXPECT 48
 EXPECT 49
 EXPECT 50
 EXPECT 51
 EXPECT 52
 EXPECT 53

```

NTOTAL = NTOTAL - IUSERS(K)                                EXPECT 54
K = K + 1                                                 EXPECT 55
IF ( K .LE. ITYPES ) GO TO 60                            EXPECT 56
70  CONTINUE                                              EXPECT 57
IUSERS1 = IUSERS + 1                                     EXPECT 58
IF (PRINTON) WRITE(6,74) NODES                           COXMOD 125
74  FORMAT(///10X,"THE MARGINAL QUEUE LENGTH PROBABILITIES ", EXPECT 60
1"AT EQUILIBRIUM"///10X,"QUEUE LENGTH",5X,"QUEUE LENGTH ", EXPECT 61
2"PROBABILITY FOR NODES 1 THROUGH ",I3)                  EXPECT 62
DO 75 I = 1, NODES                                      EXPECT 63
EXPVAL(I,1) = 0.                                         EXPECT 64
75  CONTINUE                                              EXPECT 65
DO 110 J = 1, IUSERS1                                  EXPECT 66
JJ = J - 1                                               EXPECT 67
DO 90 I = 1, NODES                                      EXPECT 68
EXPVAL(I,J) = EXPVAL(I,1) + JJ * PROBTOT(I,J)           EXPECT 69
90  CONTINUE                                              EXPECT 70
IF (PRINTON) WRITE(6,80) JJ, (PROBTOT(I,J), I=1,NODES) COXMOD 126
80  FORMAT(/14X,I3,7X,10F10.5,10(/24X,10F10.5))        EXPECT 72
110 CONTINUE                                              EXPECT 73
IF (PRINTON) WRITE(6,111)                                COXMOD 127
111 FORMAT(///10X,"THE EXPECTED NUMBER OF TASKS AT EACH NODE"///10X,
1"NODE",10X,"EXPECTED VALUE")                          EXPECT 75
DO 113 I = 1, NODES                                      EXPECT 76
IF (PRINTON) WRITE(6,112) I, EXPVAL(I,1)                 EXPECT 77
112 FORMAT(/10X,I4,10X,F10.5)                           COXMOD 128
113 CONTINUE                                              EXPECT 79
IF (PRINTON) WRITE(6,114)                                EXPECT 80
114 FORMAT(///10X,"THE MARGINAL QUEUE LENGTH PROBABILITIES ", EXPECT 82
1"AT EQUILIBRIUM CLASSIFIED BY TASK TYPE")            EXPECT 83
DO 160 J = 1, ITYPES                                    EXPECT 84
IF (PRINTON) WRITE(6,115) J, NODES                      COXMOD 130
115 FORMAT(///10X,"THE MARGINAL QUEUE LENGTH PROBABILITIES ", EXPECT 86
1"AT EQUILIBRIUM FOR TASK TYPE",I3///10X,"QUEUE LENGTH",5X, EXPECT 87
2"QUEUE LENGTH PROBABILITIES FOR NODES 1 THROUGH ",I3) EXPECT 88
IUSERS1 = IUSERS(J) + 1                                 EXPECT 89
DO 116 I = 1, NODES                                      EXPECT 90
EXPVAL(I,J) = 0.                                         EXPECT 91
116 CONTINUE                                              EXPECT 92
DO 150 K = 1, IUSERS1                                  EXPECT 93
KK = K - 1                                               EXPECT 94
DO 120 I = 1, NODES                                      EXPECT 95
EXPVAL(I,J) = EXPVAL(I,J) + KK * PROBPAR(I,J,K)        EXPECT 96
120 CONTINUE                                              EXPECT 97
IF (PRINTON) WRITE(6,130) KK, (PROBPAR(I,J,K), I=1,NODES) COXMOD 131
130 FORMAT(/14X,I3,6X,10F10.5,10(/24X,10F10.5))        EXPECT 99
150 CONTINUE                                              EXPECT 100
160 CONTINUE                                              EXPECT 101
WRITE(6,162) ITYPES                                    EXPECT 102
162 FORMAT(///10X,"THE EXPECTED NUMBER OF TASKS AT EACH NODE ",
1"CLASSIFIED BY TASK TYPE"///10X,"NODE",10X,"EXPECTED NUMBER ",
2"OF TASK TYPES 1 THROUGH ",I3," FOUND AT NODE")
DO 166 I = 1, NODES                                      COXMOD 132
166 CONTINUE                                              EXPECT 106

```

```

164 WRITE(6,164) I, (EXPVAL(I,J),J=1,ITYPES) EXPECT 107
165 FORMAT(10X,I4,10X,10F10.5,10(24X,10F10.5)) EXPECT 108
166 CONTINUE EXPECT 109
167 IF (PRINTON) WRITE(6,170) ITYPES COXMOD 133
168 FORMAT(10X,"NODE UTILIZATION BY EACH TASK TYPE"//10X,
169 1"NODE",10X,"UTILIZATION OF NODE BY TASK TYPE 1 THROUGH ",I3) EXPECT 111
170 DO 180 I = 1, NODES EXPECT 112
171 IF ( NODETYP(I) .EQ. 3 ) GO TO 180 EXPECT 113
172 IF (PRINTON) WRITE(6,175) I, (UTIL(I,J),J=1,ITYPES) EXPECT 114
173 FORMAT(10X,I4,10X,10F10.5,10(24X,10F10.5)) COXMOD 134
174 CONTINUE EXPECT 116
175 DO 190 I = 1, NODES EXPECT 117
176 DO 185 J = 2, ITYPES EXPECT 118
177 UTIL(I,1) = UTIL(I,1) + UTIL(I,J) EXPECT 119
178 CONTINUE EXPECT 120
179 CONTINUE EXPECT 121
180 IF (PRINTON) WRITE(6,195) EXPECT 122
181 FORMAT(10X,"TOTAL NODE UTILIZATION"//10X,
182 1"NODE",10X,"UTILIZATION") COXMOD 135
183 DO 200 I = 1, NODES EXPECT 124
184 IF ( NODETYP(I) .EQ. 3 ) GO TO 200 EXPECT 125
185 IF (PRINTON) WRITE(6,198) I, UTIL(I,1) COXMOD 136
186 FORMAT(10X,I4,10X,F10.5) EXPECT 129
187 CONTINUE EXPECT 130
188 IF (PRINTON) WRITE(6,210) ITYPES COXMOD 137
189 FORMAT(10X,"THE MEAN TIME EACH TASK TYPE SPENDS ",
190 1"AT THE VARIOUS NODES FOR EACH INTERACTION"//10X,"NODE",10X,
191 2"MEAN TIME FOR TASK TYPE 1 THROUGH ",I3) EXPECT 132
192 DO 230 I = 1, NODES EXPECT 133
193 IF ((DEP(I) .EQ. 1) ) GO TO 205 EXPECT 134
194 IF (PRINTON) WRITE(6,204) COXMOD 138
195 FORMAT(10X,"THE MODEL DOES NOT CALCULATE THIS TIME FOR ",/10X,
196 1"NODES WITH STATE DEPENDENT SERVICE RATES.") EXPECT 138
197 GO TO 230 EXPECT 139
198 CONTINUE EXPECT 140
199 DO 220 J = 1, ITYPES EXPECT 141
200 UTIL(I,J) = 0. EXPECT 142
201 IF ( (DEPART(I,J) .EQ. 0.) .OR. (E(I,J) .EQ. 0.) ) GO TO 220 EXPECT 143
202 UTIL(I,J) = EXPVAL(I,J) / DEPART(I,J) + E(I,J) COXMOD 145
203 CONTINUE EXPECT 146
204 IF (PRINTON) WRITE(6,225) I, (UTIL(I,J),J=1,ITYPES) COXMOD 139
205 FORMAT(10X,I4,10X,10F10.5,10(24X,10F10.5)) EXPECT 148
206 CONTINUE EXPECT 149
207 IF (PRINTON) WRITE(6,240) COXMOD 140
208 FORMAT(10X,"THE MEAN RESPONSE TIME FOR THE VARIOUS ",
209 1"TASK TYPES") EXPECT 151
210 DO 280 I = 1, NODES EXPECT 152
211 IF ( NODETYP(I) .NE. 3 ) GO TO 280 EXPECT 153
212 IF (PRINTON) WRITE(6,250) I COXMOD 141
213 FORMAT(10X,"THE MEAN RESPONSE TIME FOR TASK ",
214 1"REPRESENTED AT NODE ", I3//10X,"TASK TYPE",10X,
215 2"RESPONSE TIME") EXPECT 156
216 DO 270 J = 1, ITYPES EXPECT 157
217 EXPECT 158
218 EXPECT 159

```

```

TIME = 0.
IF ( DEPART(I,J) .EQ. 0.) GO TO 255
TIME = (IUSERS(I,J) - EXPVAL(I,J)) / DEPART(I,J)
255 CONTINUE
IF (PRINTON) WRITE(6,260) J, TIME
260 FORMAT(13X,I3,14X,F10.5)
270 CONTINUE
280 CONTINUE
RETURN
C
C THAT'S ALL FOLKS!!
C
END
SUBROUTINE CHEN75(PROB,NDISKS,DISKPRB,TCPU,DIO,NAVJOB,
1EXPVAL,SERVICE,DELTA,PEXPVAL,IUSERS,MAXNODE,MAXTYPE,
2TYPES,NODES,NITER,E,X,ZAREA)
DIMENSION PROB(MAXNODE,MAXNODE),DISKPRB(MAXNODE,MAXTYPE),
1DIO(MAXTYPE),EXPVAL(MAXNODE,MAXTYPE),SERVICE(MAXNODE,MAXTYPE),
2E(MAXNODE,MAXTYPE),X(MAXNODE),TCPU(MAXTYPE),
2ZAREA(MAXNODE)
REAL DELTA,PEXPVAL
INTEGER NDISKS,MAXNODE,MAXTYPE,IUSERS
C
C THIS SUBROUTINE CALCULATES THE DELTA BETWEEN THE LAST
C ITERATIONS ANSWER FOR THE NUMBER OF JOBS IN THE "THINK STATE"
C AND THE CURRENT ITERATIONS ANSWER CONTAINED IN
C EXPVAL(NDISKS+3,1). ALSO, USING THE SUBROUTINE COMPUTE IT
C RECOMPUTES THE PROBABILITY TRANSITION MATRIX, PROB, USING THE
C FORMULAE CONTAINED IN CHEN'S ARTICLE, "QUEUEING NETWORK MODEL
C OF INTERACTIVE COMPUTING SYSTEMS", IN THE
C PROCEEDINGS OF THE IEEE, VOL. 63, NO. 6, JUNE 75.
C
WRITE(6,67) NITER
67 FORMAT(1H1,///10X,"TRANSITION PROBABILITIES DESCRIBING",
1"MOVEMENT OF TASKS AMONG THE MODEL'S NODES FOR ITERATION ",I5)
DO 110 K = 1, ITYPES
WRITE(6,68) K, NODES
68 FORMAT(//10X,"TRANSITION PROBABILITIES FOR TASK TYPE ",
113//10X,"DEPARTURE",7X,"PROBABILITY OF TASK MOVEMENT FROM"/12X,
2"NODE",9X,"DEPARTURE NODE TO NODES 1 THROUGH ",I3)
CALL COMPUTE(K,PROB,NDISKS,DISKPRB,TCPU,DIO,
1EXPVAL,SERVICE,IUSERS,MAXNODE,MAXTYPE,NODES,NAVJOB)
IF (NITER .NE. 0) GO TO 70
PROB(1,1) = PROB(1,1) + PROB(NDISKS+2,1)
PROB(NDISKS+2,1) = 0.0
70 CONTINUE
DO 73 I = 1, NODES
WRITE(6,74) I, (PROB(J,I),J=1,NODES)
74 FORMAT(/12X,14,10X,10F10.5,10(1/24X,10F10.5))
75 CONTINUE
NUMPROB = MAXNODE + MAXNODE

```

EXPECT 160
 EXPECT 161
 EXPECT 162
 EXPECT 163
 COXMOD 142
 EXPECT 165
 EXPECT 166
 EXPECT 167
 EXPECT 168
 EXPECT 169
 EXPECT 170
 EXPECT 171
 EXPECT 172
 COXMOD 143
 COXMOD 144
 COXMOD 145
 COXMOD 146
 COXMOD 147
 COXMOD 148
 COXMOD 149
 COXMOD 150
 COXMOD 151
 COXMOD 152
 COXMOD 153
 COXMOD 154
 COXMOD 155
 COXMOD 156
 COXMOD 157
 COXMOD 158
 COXMOD 159
 COXMOD 160
 COXMOD 161
 COXMOD 162
 COXMOD 163
 COXMOD 164
 COXMOD 165
 COXMOD 166
 COXMOD 167
 COXMOD 168
 COXMOD 169
 COXMOD 170
 COXMOD 171
 COXMOD 172
 COXMOD 173
 COXMOD 174
 COXMOD 175
 COXMOD 176
 COXMOD 177
 COXMOD 178
 COXMOD 179
 COXMOD 180
 COXMOD 181
 COXMOD 182

```

CALL WRITNS(3,PROB(1,1),NUMPROB,1) COXMOD 183
DO 80 I = 1, NODES COXMOD 184
PROB(I,I) = PROB(I,I)-1. COXMOD 185
PROB(1,1) = 0. COXMOD 186
K(I) = 0. COXMOD 187
-80 CONTINUE COXMOD 188
PROB(1,1) = 1. COXMOD 189
X(1) = 1. COXMOD 190

C COXMOD 191
C THE RELATIVE TRANSITION RATE BETWEEN NODES IS CALCULATED COXMOD 192
C SOLVING THE SET OF LINEAR EQUATIONS SPECIFIED BY PROB AND X. COXMOD 193
C THE SUBROUTINE LEQTIF SOLVES THIS SET OF LINEAR EQUATIONS. COXMOD 194
C LEQTIF IS A SUBROUTINE FROM THE IMSL PACKAGE SUPPORTED AT COXMOD 195
C WRIGHT-PATTERSON AFB, OHIO. COXMOD 196
C COXMOD 197

CALL LEQTIF(PROB,1,NODES,MAXNODE,X,0,WAREA,IER) COXMOD 198
CALL READNS(3,PROB(1,1),NUMPROB,1) COXMOD 199
DO 100 I = 1, NODES COXMOD 200
GAMMA = 0. COXMOD 201
DO 90 J = 1, NODES COXMOD 202
GAMMA = GAMMA + (PROB(I,J) * X(J)) COXMOD 203
90 CONTINUE COXMOD 204
IF ( ABS(GAMMA - X(I)) .LE. .1 ) GO TO 100 COXMOD 205
WRITE(6,95) I, (X(L),L=1,NODES) COXMOD 206
95 FORMAT(10X,"THE RELATIVE SERVICE RATE BETWEEN NODES CALCULATED ", COXMOD 207
1"FROM THE COEFFICIENT ARRAY PROB IS IN ERROR FOR NODE ", COXMOD 208
2I3//5X,"THE RELATIVE SERVICE RATES ARE ",/10X, COXMOD 209
3I0/10(F10.5,2X)) COXMOD 210
GO TO 113 COXMOD 211
100 CONTINUE COXMOD 212
DO 105 I= 1, NODES COXMOD 213
E(I,K) = X(I) COXMOD 214
105 CONTINUE COXMOD 215
110 CONTINUE COXMOD 216
115 CONTINUE COXMOD 217
RETURN COXMOD 218
END COXMOD 219

SUBROUTINE COMPUTE(JCLASS,PROB,NDISKS,DISKPRB,TCPU,DIO,
1EXPVAL,SERVICE,IUSERS,MAXNODE,MAXTYPE,NODES,NAVGJOB) COXMOD 220
DIMENSION PROB(MAXNODE,MAXTYPE),DISKPRB(MAXNODE,MAXTYPE),
1DIO(MAXTYPE),EXPVAL(MAXNODE,MAXTYPE),
2SERVICE(MAXNODE,MAXTYPE),TCPU(MAXTYPE) COXMOD 221
INTEGER JCLASS,NDISKS,IUSERS,MAXNODE,MAXTYPE,ITYPES COXMOD 222
C COXMOD 223
C THIS SUBROUTINE COMPUTES THE PROBABILITY TRANSITION MATRIX COXMOD 224
C FOR THE MODEL BASED ON THE INPUTS OF CHEN'S MODEL (NDISKS, DISKPRB, COXMOD 225
C TCPU, AND DIO) AND HIS FORMULAE. COXMOD 226
C COXMOD 227
C INITIALIZE PROBABILITY TRANSITION MATRIX COXMOD 228
DO 10 I = 1, NODES COXMOD 229
DO 10 J = 1, NODES COXMOD 230
IF (J .EQ. 1) GO TO 5 COXMOD 231
PROB(J,I) = 0.0 COXMOD 232
10 CONTINUE COXMOD 233
IF (J .EQ. 5) GO TO 10 COXMOD 234
PROB(J,I) = 1.0 COXMOD 235

```

```

GO TO 10                                COXMOD 236
5   PROB(J,I) = 1.0                      COXMOD 237
10  CONTINUE                               COXMOD 238
    PROB(NDISK5+3,1) = SERVICE(1,JCLASS) / TCPU(JCLASS)
    DO 20 I = 1, NDISKS                   COXMOD 239
    PROB(I+1,1) = DID(JCLASS) + PROB(NDISK5+3,1) + DISKPRB(I,JCLASS) COXMOD 240
20  CONTINUE                               COXMOD 241
    IF ((EXPVAL(NDISK5+3,1) .GE. IUSERS - NAVGJOB) .AND.
        1(IUSERS .GE. NAVGJOB))           COXMOD 242
    2PROB(NDISK5+2,NDISK5+3) = (IUSERS - NAVGJOB) / EXPVAL(NDISK5+3,1) COXMOD 243
                                                COXMOD 244
    IF ( IUSERS .LE. NAVGJOB ) PROB(NDISK5+2,NDISK5+3) = 0.0          COXMOD 245
                                                COXMOD 246
    IF ( EXPVAL(NDISK5+3,1) .LE. (IUSERS - NAVGJOB) )
1      PROB(NDISK5+2,NDISK5+3) = 1.0          COXMOD 247
                                                COXMOD 248
    PROB(1,NDISK5+3) = 1.0 - PROB(NDISK5+2,NDISK5+3)                  COXMOD 249
                                                COXMOD 250
    CHOICE1 = 0.0                          COXMOD 251
    CHOICE2 = 0.0                          COXMOD 252
    ACTIVE = EXPVAL(1,1) + EXPVAL(NDISK5+2,1)                  COXMOD 253
    DO 40 I = 1, NDISKS                   COXMOD 254
    ACTIVE = ACTIVE + EXPVAL(I+1,1)          COXMOD 255
    CHOICE1 = CHOICE1 + PROB(I+1,1)          COXMOD 256
40  CONTINUE                               COXMOD 257
    CHOICE1 = CHOICE1 + PROB(NDISK5+3,1)      COXMOD 258
    CHOICE1 = 1.0 - CHOICE1                 COXMOD 259
    IF ((ACTIVE .LT. NAVGJOB) .OR. (IUSERS .LE. NAVGJOB)) GO TO 50
    CHOICE2 = PROB(NDISK5+3,1) * (ACTIVE - NAVGJOB) /
1 (IUSERS - NAVGJOB)                      COXMOD 260
    PROB(NDISK5+2,1) = AMIN1(CHOICE1,CHOICE2)      COXMOD 261
    GO TO 60                                COXMOD 262
50  PROB(NDISK5+2,1) = 0.0                  COXMOD 263
60  PROB(1,1) = 1.0 - PROB(NDISK5+2,1) - (1.0 - CHOICE1)      COXMOD 264
    RETURN                                 COXMOD 265
    END                                    COXMOD 266
*END
*EOF
}

```

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GOR/MA/81D-4	2. GOVT ACCESSION NO. <i>0115 565</i>	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MULTI-CLASS ANALYTICAL MODELS OF THE DECSYSTEM-10 JOB-SWAPPING BEHAVIOR	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis	
7. AUTHOR(s) Michael H. Cox Capt USAF	6. PERFORMING ORG. REPORT NUMBER	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE December 1981	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	13. NUMBER OF PAGES 150	
	15. SECURITY CLASS. (of this report) Unclassified	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) <i>15 APR 1982</i>		
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17 <i>John F. Walawender</i> FREDERIC C. LYNCH, Major, USAF Director of Public Affairs	Dean for Research and Professional Development Air Force Institute of Technology (ATC) Wright-Patterson AFB, OH 45433	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Closed Queueing Network Models Computer Performance Evaluation Time-sharing Computer Systems Approximate Solutions to Queueing Networks		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) An improved model of the DECSYSTEM-10 job-swapping behavior was developed. This model combines a previously developed closed queueing network model with a job-swapping model developed by Chen (Ref 5). Chen's swapping model provides an approximate solution to a network queueing model with a state-dependent probability transition matrix. This combined model is then tested on a hypothetical, though realistic workload containing both interactive and batch jobs. The two classes of jobs are treated first as separate classes, as one class having		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 68 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) A-20

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

just the interactive job characteristics. The results of these experiments and a comparison between Chen's swapping model and the classical are presented.

The results of the experiments indicate that it is important to model multiple classes for systems which have a significant amount of batch activity. Also, Chen's swapping model provides a more realistic model of job-swapping behavior for the DECsystem-10. Therefore, combining the multi-class model with Chen's swapping model improves the modeling accuracy for the DECsystem-10. Recommendations for extensions to this multi-class Chen model are also discussed.

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

A-21

**DATE
TIME**